零起点学自动化技术丛书

零起点

学Proteus单片机仿真技术

LINGQIDIAN XUE PROTEUS DANPIANJI FANGZHEN JISHU

范海绍 李方园 等编著



机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE Caliente.COM

● 零起点学自动化技术丛书 ●

零起点学 Proteus 单片机仿真技术

范海绍　李方园　等编著



机 械 工 业 出 版 社

本书介绍了用 Proteus 仿真软件工具进行单片机应用设计的基本方法，深入浅出地讲解了 Proteus 的常用功能、Keil C51 软件编译和单片机应用电路设计和仿真，并详细地讲解了常用单片机功能电路的设计方法和仿真，使读者在阅读和实验中体会 Proteus 强大的功能和学习单片机应用的快乐。

本书适合对电子设计有浓厚兴趣的初级读者，也适合作为高职院校开设的单片机技术课程和各种单片机技术培训班教材使用。

免费下载案例源程序，网址：http//www. cmpbook. com

图书在版编目（CIP）数据

零起点学 Proteus 单片机仿真技术/范海绍等编著. —北京：机械工业出，版社2012. 1(2015.1重印)

（零起点学自动化技术丛书）

ISBN 978-7-111-36904-2

Ⅰ. ①零… Ⅱ. ①范… Ⅲ. ①单片微型计算机 – 系统设计 – 应用软件，PROTEUS Ⅳ. ①TP368. 1

中国版本图书馆 CIP 数据核字（2011）第 270886 号

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）

策划编辑：林春泉　责任编辑：吕　潇

版式设计：霍永明　责任校对：申春香

封面设计：路恩中　责任印制：刘　岚

北京富生印刷厂印刷

2015 年 1 月第 1 版第 2 次印刷

184mm × 260mm · 11. 5 印张 · 281 千字

0001—4000册

标准书号：ISBN 978-7-111-36904-2

定价：38.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务　网络服务

社 服 务 中 心：（010）88361066　门户网：http：//www. cmpbook. com

销　售　一　部：（010）68326294　教材网：http：//www. cmpedu. com

销　售　二　部：（010）88379649

读者购书热线：（010）88379203

封面无防伪标均为盗版

前　言

随着电子技术、计算机软件技术的飞速发展，电子设计方法也在不断进步。Proteus 嵌入式虚拟开发系统和仿真平台给我们提供了前所未有强大的开发环境，使我们在开始制作硬件之前就能进行设计和验证产品的性能、功能、界面等特性，从设计原理图、软件设计，到系统仿真一气呵成。它避免了传统电子电路设计中方案更换带来的多次重复购买元器件及制板、焊接、软件调试等麻烦，节省了大量的时间和经费，提高了设计效率和质量。同时也为初学者打开了方便之门，降低了学习门槛，更加有利于电子设计技术的普及和提高。

俗话说，好的开始是成功的一半。学习用 Proteus 进行单片机应用设计，将使读者体会电子设计的无比轻松和成功的喜悦，从而激发读者的学习热情和创造灵感。

本书采用即学即用的教学方法，帮助读者把学到的知识和技巧立刻变成可用的技术，而不是一味系统地介绍软件的功能、追求知识的完整性。这里我们运用 20/80 原则：只讲 Proteus 中 20% 的功能，用它实现 80% 的设计，余下的 80% 内容读者完全能够通过自学和今后的工作中不断完善和加强。

单片机应用设计涉及很多领域，包括模拟电子、数字电子、软件设计等技术，每一个技术领域都是一个庞大的、系统的知识领域，我们不可能在完全熟悉了以上这些知识后再来学习单片机应用设计，但是不可否认，精通以上技术领域一定能更有效地进行单片机应用设计。在涉及某个领域技术时，我们先采用“拿来主义”，把它用在设计中，以后再慢慢消化。全才的人毕竟很少，我们每个人往往擅长某个方面，没有必要追求“十全十美”，可以采用合作的方式设计开发产品，所以，“拿来主义”就是一种合作开发的方式。随着社会的进步和社会化大生产的发展，社会分工会越来越细，我们应该学会采用别人的成果为自己所用，即站在巨人的肩膀上，才能看得更远，飞得更高。

本书首先介绍 Proteus 的安装、使用基本方法，然后逐步引出单片机的应用设计，将单片机所涉及的外围设备逐一用案例介绍给读者，每个案例都会包括单片机的功能介绍、电路设计、软件设计和系统仿真。并通过一个综合案例将几个外部设备的应用串起来使用，最后给出两个比较大的应用案例，进一步体现 Proteus 设计的方便性和强大功能。最后一讲介绍制版的方法，将设计付诸实现，圆满地完成设计任务，交付生产。另外，Proteus 软件中使用的元器件图形和文字符号有些与我国现行标准不一致，为维持计算机显示的原貌，本书并未刻意对此进行统一，请读者在阅读时加以分别。

本书参加编写的还有杨帆、乐斌、钟晓强、邵振翔、蔡美茹、陈亚玲、叶明、方定桂、戴琴、王永行、刘军毅等，本书在编写过程中参考了国内最新出版的各类文献资料，在此向这些文献资料的作者一并致谢！

作　者

2011 年 11 月

目　　录

前言
第 1 讲　Proteus 基本操作 …… 1
1.1　Proteus 概况 …… 1
1.2　Proteus 软件包安装 …… 2
1.3　ISIS 设计环境简介 …… 9
1.4　ISIS 电路设计元器件查找 …… 10
1.5　ISIS 电路设计与仿真：第一个电路设计 …… 12
第 2 讲　元器件属性及编号设置 …… 19
2.1　元器件的摆放 …… 19
2.2　电路连线的编辑 …… 24
2.3　设置电源属性 …… 25
2.4　设置元器件属性 …… 26
2.5　用工具快速设置元器件属性 …… 28
第 3 讲　单片机程序编写和编译 …… 32
3.1　Keil C51 使用方法 …… 32
3.2　第一个 C 语言程序：单片机点亮一个发光二极管 …… 37
3.3　设计电路和仿真运行程序 …… 39
第 4 讲　节点、总线连接技术 …… 43
4.1　单片机流水灯控制电路 …… 43
4.2　用总线连接发光二极管至单片机 …… 47
4.3　程序设计和仿真运行 …… 49
第 5 讲　1 位数码管计数器 …… 55
5.1　1 位数码管计数电路 …… 55
5.2　按钮与中断处理 …… 58
5.3　程序设计与仿真 …… 60
第 6 讲　4 位数码管计时器 …… 65
6.1　4 位数码管计时器电路 …… 65
6.2　多位数码管显示程序设计 …… 66
6.3　定时器原理和程序设计 …… 68
6.4　程序设计与仿真 …… 73
第 7 讲　蜂鸣器 …… 77
7.1　蜂鸣器应用电路及发声原理 …… 77
7.2　程序设计与仿真 …… 79
第 8 讲　继电器 …… 82

8.1 继电器驱动电路 …… 82
8.2 程序设计与仿真 …… 85
第9讲 定时供电插座设计 …… 87
9.1 定时供电插座需求分析 …… 87
9.2 定时供电插座电路设计 …… 88
9.3 程序设计与仿真 …… 94
第10讲 RS232串口通信 …… 103
10.1 最简单的单片机串口通信电路 …… 104
10.2 通过MAX232转换的串口通信电路 …… 105
10.3 多机串口通信电路 …… 106
10.4 程序设计与仿真 …… 108
10.5 PC（虚拟终端）与单片机通信 …… 110
第11讲 RS485串口通信 …… 119
11.1 RS485串口通信电路 …… 119
11.2 多机RS485串口通信电路 …… 120
11.3 程序设计与仿真 …… 121
第12讲 A-D转换 …… 127
12.1 A-D转换原理和电路设计 …… 127
12.2 程序设计与仿真 …… 129
第13讲 并口扩展 …… 133
13.1 8255A可编程并行I/O接口芯片 …… 133
13.2 并口扩展电路设计 …… 135
13.3 程序设计与仿真 …… 136
第14讲 步进电动机控制 …… 138
14.1 步进电动机控制原理 …… 138
14.2 步进电动机控制电路 …… 141
14.3 程序设计与仿真 …… 141
第15讲 直流电动机控制 …… 147
15.1 直流电动机控制原理 …… 147
15.2 用定时器控制占空比 …… 147
15.3 用示波器查看占空比 …… 151
15.4 直流电动机控制电路 …… 153
15.5 程序设计与仿真 …… 154
第16讲 交通指挥灯控制 …… 155
16.1 红绿灯控制需求分析 …… 155
16.2 红绿灯控制电路设计 …… 156
16.3 程序设计与仿真 …… 157
第17讲 水箱水位控制 …… 161
17.1 水箱水位控制需求分析 …… 161

17.2 水箱水位控制电路设计 …… 162
17.3 程序设计与仿真 …… 163
第 18 讲 Proteus 电路制板 …… 166
18.1 ARES 电路制板简介 …… 166
18.2 电路制板操作步骤和实例 …… 167
参考文献 …… 176

第1讲 Proteus 基本操作



导读

本讲将介绍 Proteus 的概况、Proteus 的安装方法、Proteus 电路设计 ISIS 环境简介，以及 ISIS 环境下电路设计的元器件选择。在此基础上进行 ISIS 环境下第一个电路设计与仿真：电源、电灯与开关。

1.1 Proteus 概况

Proteus 软件是由英国 Labcenter Electronics 公司开发的 EDA（Electronic Design Automation，电子设计自动化）工具软件，它集成了高级原理图布图、混合模式 SPICE 电路仿真、PCB（Printed Circuit Board，印制电路板）设计以及自动布线，可实现一个从概念产品到设计完成的完整的电子设计。它由 ISIS 和 ARES 两个软件构成，其中 ISIS 是一款便捷的电子系统仿真平台软件，ARES 是一款高级的布线编辑软件。

Proteus 产品系列包含 VSM 技术，用户可以对基于微控制器的设计连同所有周围电子元器件一起仿真，甚至可以实时采用诸如 LED/LCD、键盘、RS232 终端等动态外设模型来对设计进行仿真。

ISIS 是 Proteus 系统的中心，它提供给用户图形外观包括线宽、填充类型、字符等全部控制，使用户能够生成精美的原理图，原理图可以以图形文件的格式输出，或者复制到剪贴板以便其他文件使用。这就使得 ISIS 成为制作技术文件、学术论文、项目报告的理想工具，也是 PCB 设计的一个出色前端。

ISIS 使最常用的画图操作变得又快又容易，布置、编辑、移动和删除操作能够用鼠标直接实现，无需去单击菜单或图标。单击想要连接的两个引脚，就能简单地实现布线。自动布线也能在元器件移动时操作，自动地解决相应的连线、节点自动布置和移除问题。

和支持通常的多图样设计过程一样，ISIS 支持层次设计。特殊的元器件能够定义为通过电路图表示的模块，能够任意设定层次。

ISIS 提供总线支持，不仅是一根总线，还能用总线引脚定义元器件和子电路。因此，一个连接在处理器和存储器之间的 32 位处理器总线可以用单一的线表示，节省绘图的时间和空间。

Proteus 软件的模拟仿真功能直接兼容厂商的 SPICE 模型，采用扩充的 SPICE3F5 的电路仿真模型，能够记录基于图表的频率特性、直流电的传输特性、参数的扫描、噪声分析、傅里叶分析等，具有超过 8 000 种的电路仿真模型，包括标准符号、晶体管、二极管、热离子管、TTL、CMOS、ECL、微处理器以及存储器部件、PLD、模拟 IC 和运算放大器。

Proteus 软件的数字仿真支持 JDEC 文件的物理器件仿真，有全系列的 TTL 和 CMOS 数字仿真模型，同时一致性分析易于系统的自动测试。支持许多通用的微控制器，如 PIC、AVR、HC11 以及 8051、ARM7 等；包含强大的调试工具，可对寄存器、存储器实时监测，具有断点调试功能及单步调试功能；具有对显示器、按钮、键盘等外设进行交互可视化仿真。此外，Proteus 可对 IAR C-SPY、Keil 等开发工具的源程序进行调试，可与 Keil 实现联调。

Proteus 中的整个电路仿真是在 ISIS 原理图设计模块下延续下来的，原理图中，曲线图和电路激励以及直接布置在线路上的探针一起出现在元器件的旁边，在任何时候都能通过按下空格键对电路进行仿真，加快从编辑到仿真的速度。仿真器有独自的应用窗口和用户界面。

在传统的基于曲线图的地名录仿真的基础上，Proteus VSM 提供了完全交互电路动画曲线。用户能够用鼠标操作元器件模型控制设计，并能够从指示屏上观察到过程。此外提供了很多虚拟仪器，如电压计、电流计、示波器等 14 种虚拟仪器。这些虚拟仪器使得电路仿真非常直观，如同在实际中操作一样。

用户可以用 Proteus 卓越的建模工具，创建自己的元器件模型，ISIS 中支持层次化设计，使用户能够创建虚拟的测试步骤来开发元器件模型。用户也可以使用 VSM API 在 Windows DLL 里用 C + + 等编程语言实现模拟和数字模型，VSM API 也可以用于实现复杂的动画器件。

ARES 则代表了最复杂的 PCB 设计技术，在最大规格为 20m 的板内，布置分辨率为 10nm，支持 16 个铜箔层，2 个丝印层，4 个机械层加上板沿，禁止布线层，抗蚀掩膜和阻焊层。支持任意角元器件的布置和焊盘栈，完全自动的连线以及力矢量生成，是理想的基于网表的手工布线系统，物理设计规则检测保证设计的完整性，超过 1000 种标准封装元器件库，完整的 CAM 输出。

1.2 Proteus 软件包安装

我们以 Proteus7.6 版本为例介绍安装的具体方法。首先，打开软件包，找到安装启动程序 setup.exe，并用鼠标左键双击该文件，按照提示进行安装，如图 1-1 所示。

用鼠标左键单击“Next”键继续，如图 1-2 所示。

用鼠标左键单击“Yes”键继续，如图 1-3 所示。

用鼠标左键单击“Next”键继续，如图 1-4 所示。

用鼠标左键单击“Next”键继续，如图 1-5 所示。

用鼠标左键单击“Browse For Key File”键，浏览查找 Licence 文件，找到软件包中“LICENCE.lxk”文件，找到 Licence 文件后，然后用鼠标左键单击“Install”键，用鼠标

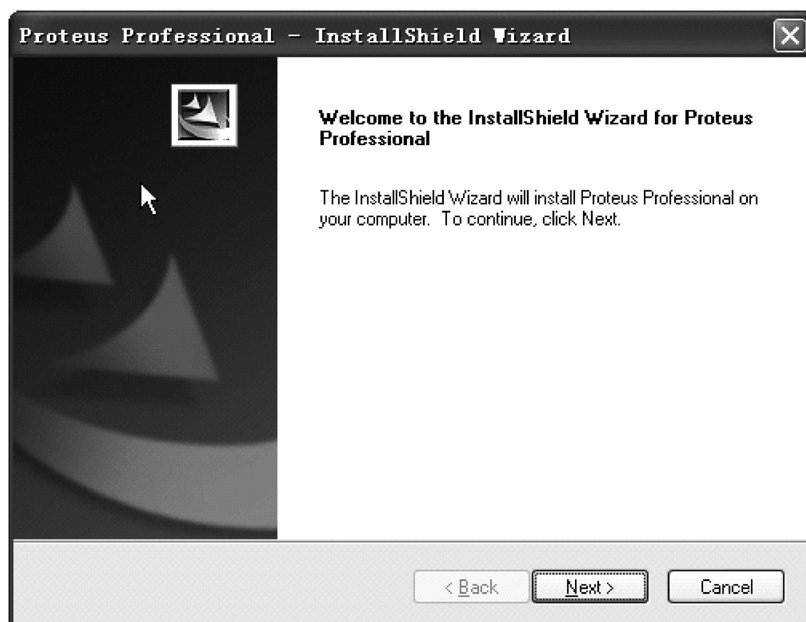


图 1-1　安装欢迎界面



图 1-2　软件协议阅读确认

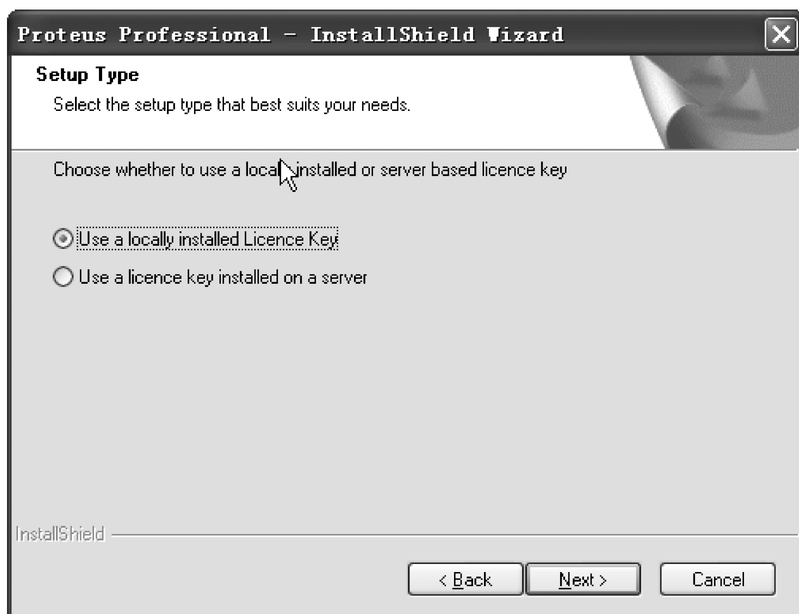


图 1-3　Licence 查找位置确认

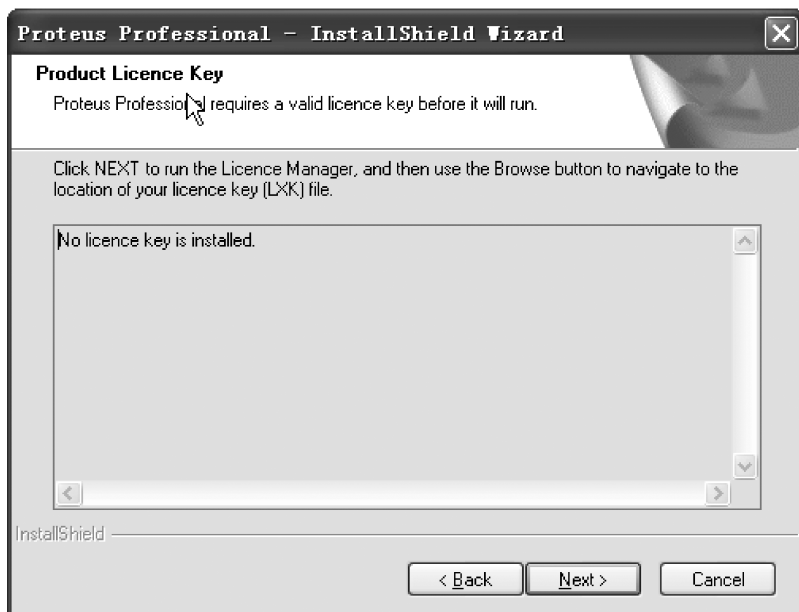


图 1-4　Licence 显示

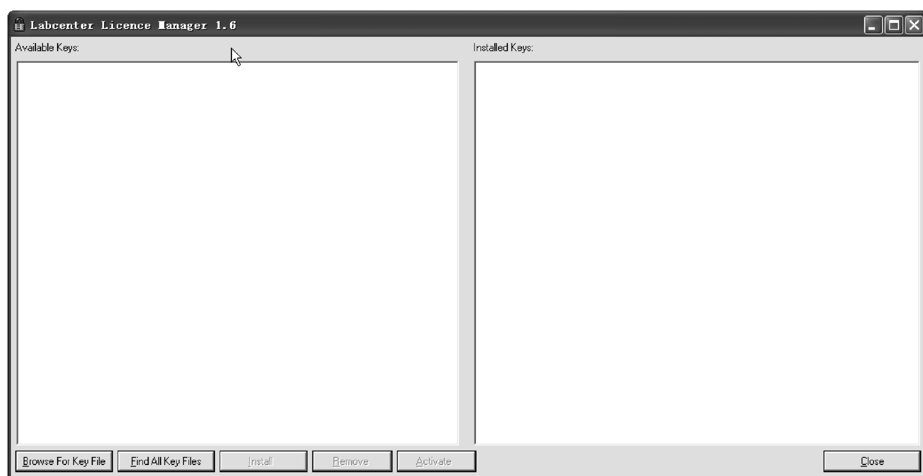


图 1-5 查找 Licence 对话框

左键单击“是（Y）”键继续，用鼠标左键单击“Close”键关闭该窗口，弹出 Licence 显示窗口，用鼠标左键单击“Next”键继续。

若需改变安装路径，用鼠标左键单击“Browse”键选择新的安装路径，否则用鼠标左键单击“Next”键继续，如图 1-6 所示。



图 1-6 选择安装路径

在安装选项中选择希望的选项，用鼠标左键单击“Next”键继续，如图 1-7 所示。

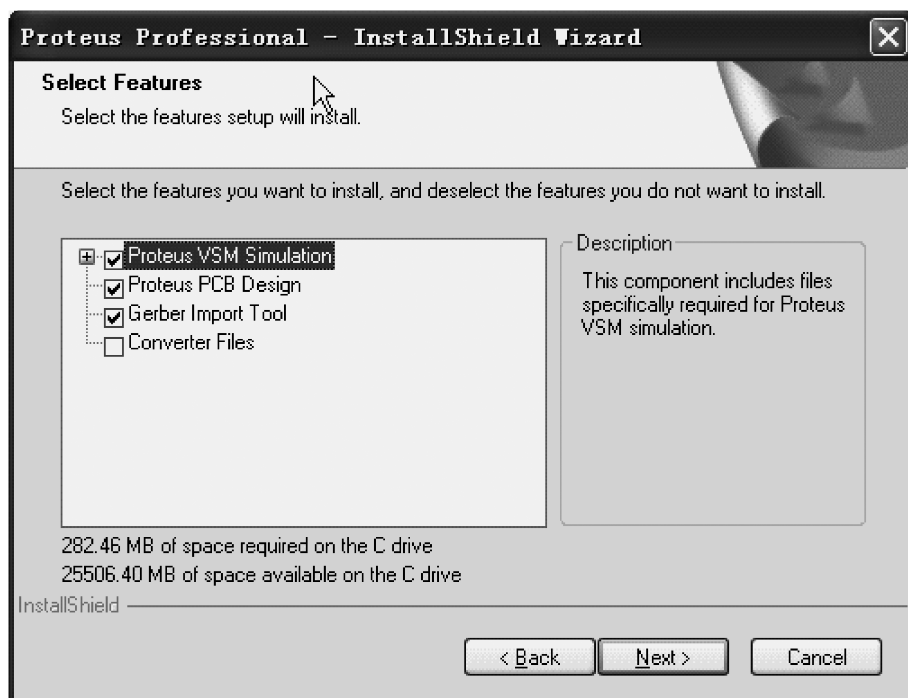


图 1-7　安装选项选择

输入你想要的程序名称，用鼠标左键单击“Next”键继续，如图 1-8 所示。



图 1-8　程序目录和名称选择

程序开始安装，图 1-9 所示为程序安装界面。



图 1-9　程序安装界面

等待程序安装完成，弹出完成对话框，安装完成后，可以选择希望查看的文件，并用鼠标左键单击“Finish”键继续，如图 1-10 所示。若选择查看 README 文件，就会弹出 README 文件，如图 1-11 所示。

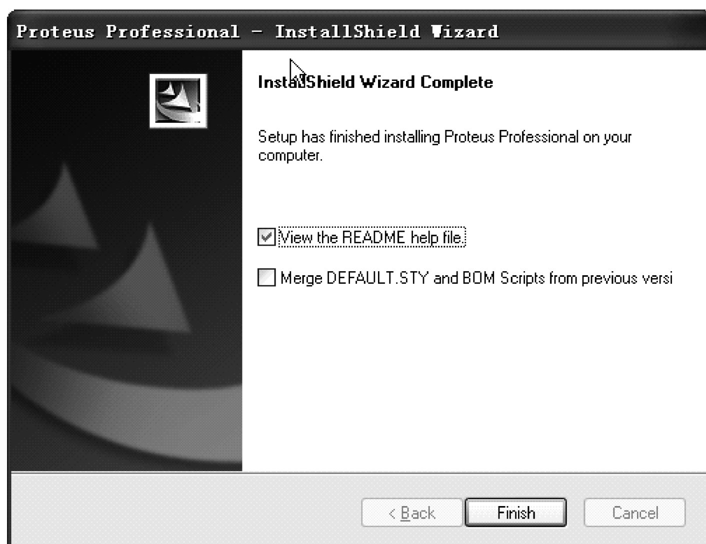


图 1-10　安装完成对话框

然后，需要将汉化包复制到安装目录中，默认安装在 C：\Program Files\Labcenter Electronics\ Proteus 7 Professional 目录下，将汉化包中的 2 个文件（见图 1-12）复制到安装目录的 bin 目录中，这样，安装就大功告成了。接下来我们就能启动 Proteus 中的 ISIS 程序，进入 ISIS 设计界面（见图 1-13），开始设计之旅。

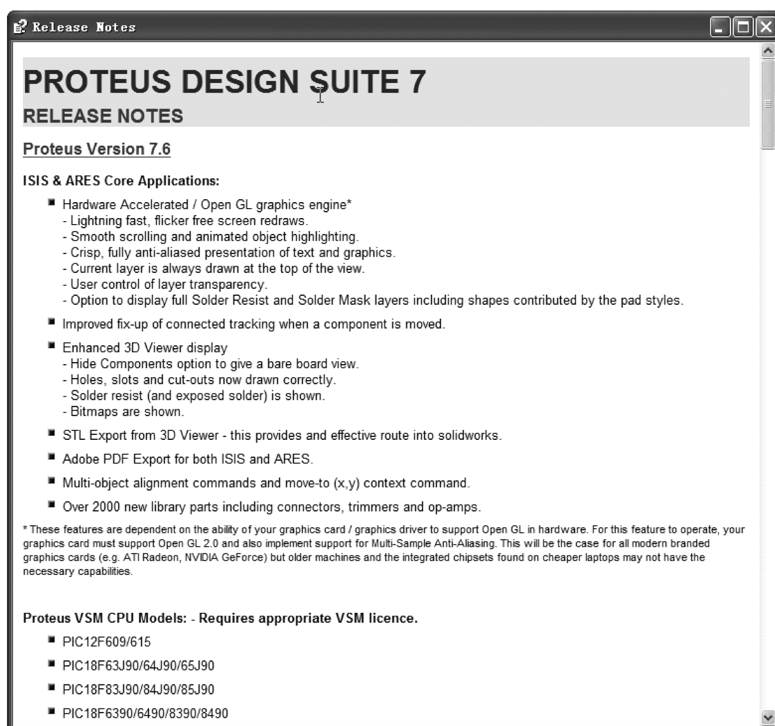


图 1-11　显示 README 文件



图 1-12　汉化包中 2 个文件

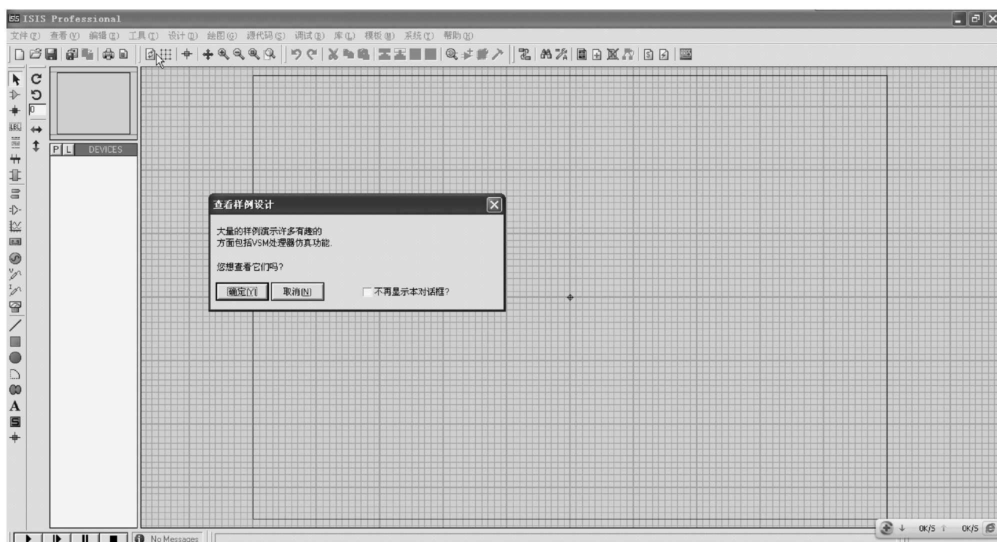


图 1-13　ISIS 启动界面

1.3 ISIS 设计环境简介

下面我们开始介绍 ISIS 设计环境，确切地说应该是 Proteus 的 ISIS 7 Professional 界面介绍。启动 ISIS 菜单或图标，我们看到如图 1-14 所示的窗口。



图 1-14 ISIS 编辑窗口介绍

1）原理图编辑窗口：顾名思义，它是用来绘制原理图的。蓝色方框内为可编辑区。这个窗口没有滚动条，可以用预览窗口来改变原理图的可视范围，也可用鼠标配合鼠标的滚动轮改变可视范围。

2）预览窗口：它可以显示两个内容，一个是当你在元器件列表中选择一个元器件时，它显示该元器件的预览图；另一个是当你的鼠标焦点落在原理图编辑窗口时，它会显示整张原理图的缩略图，并会显示一个绿色的方框，里面的内容就是当前原理图窗口中显示的内容，因此，你可用鼠标在它上面单击来改变绿色方框的位置，从而改变原理图的可视范围。

3）元器件列表窗口：用于存放元器件。当你用鼠标左键单击“P”按钮会打开元器件对话框，选择了一个元件或器件后，该元（器）件就会在元器件列表中显示，以后要用到该元（器）件时，只需在元器件列表中选择即可。

4）模型选择工具栏：用于选择不同模型，即决定元器件列表窗口出现的类型。如选择元器件模型，元器件列表窗口就列出你所选的元器件列表。若选择终端接口模型，元器件列

表窗口就出现各种终端接口，如 VCC、地、输入接口、输出接口等。若选择虚拟仪器模型，元器件列表窗口就会列出各种虚拟仪器的名字供选择等。

5）元器件方向工具栏：用于选择或调整元器件的摆放方向。例如，原来电阻显示的是水平摆放位置，通过旋转 90°，使它变成垂直摆放位置。共有 4 种选择，分别是左旋和右旋 90°的整倍数，以及水平和垂直翻转。

6）仿真按钮：用于执行电路的仿真，共 4 个按钮，分别是“运行”、“单步运行”、“暂停”和“停止”。

1.4 ISIS 电路设计元器件查找

有了对 ISIS 设计环境的基本了解后，我们开始着手进入设计环节。首先，需要了解如何选择元器件，因为元器件数量很大，要随时找到我们所要的元器件并非易事。在设计前，一般先把需要的元器件都找出来，放在前面介绍的“元器件列表窗口”中（当然也可以临时添加），然后从元器件列表窗口把元器件搬到“原理图编辑窗口”。

用鼠标左键单击 Proteus 的 ISIS 7 Professional 菜单或图标，进入 ISIS 设计环境。用鼠标左键单击“元器件列表窗口”的左上角的“P”按钮，就会弹出一个“Pick Devices”对话框，如图 1-15 所示。



图 1-15 查找元器件对话框

选择元器件有以下 3 种方法：

1. 直接输入元器件名称查找

在关键字（Keywords）文本框中直接输入你想查找的元器件名称，大小写无关，若该元

器件存在，就会在中间的结果（Results）框中出现，然后用鼠标左键双击出现在框中元器件的名称，将该元器件放入“元器件列表窗口”。

例如，查找一个标准的电阻，名称是“RES”，操作方法如图 1-16 所示。

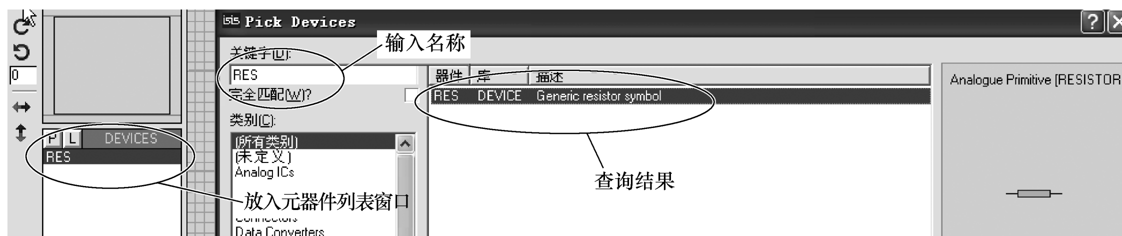


图 1-16　查找电阻元件

2. 通过类别查找

可以通过类别（Category）、子类别（Sub category）和制造商（Manufacture）来选择元器件，通过这种方法，可以查找到非常具体的元器件。例如，类别是“Resistors”，子类别是 0.6W Metal Film，制造商是 Maplin，如图 1-17 所示。

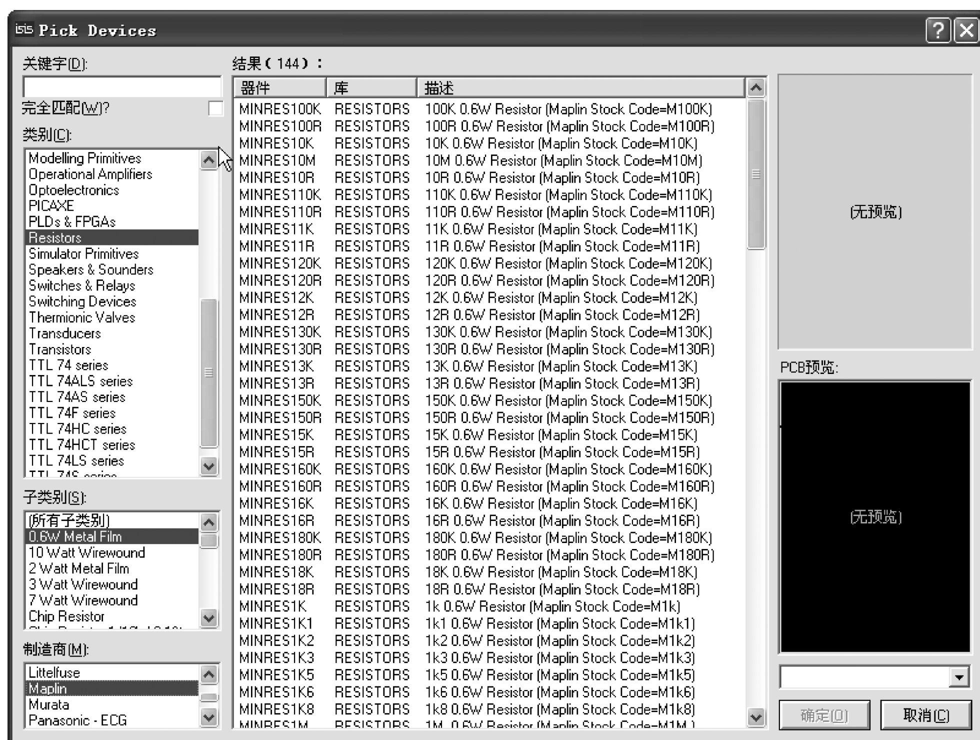


图 1-17　通过类别查找

若不知道子类别应该选什么，可以选择所有子类别（All Sub category），但有时可能会因为符合条件的元器件太多而不能显示。同样，制造商可以选择所有制造商（All Manufacture）。

3. 通过某几项参数值查找

有时候我们只知道该元器件中的某几项参数值和部分名称，也可以很方便地查找，只要将你知道的值逐一输入关键字（Keywords）框，用空格分隔，就能搜寻到相关的元器件。例如，在 Keywords 框中输入以下参数值：7805 30V 13A 2.5W，如图 1-18 所示。

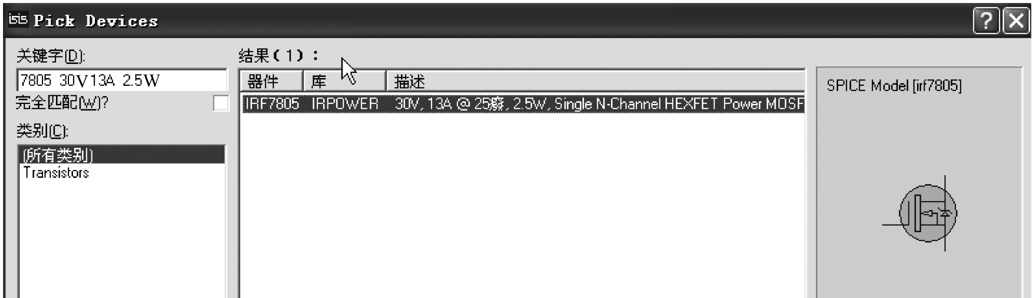


图 1-18　按关键字查找

1.5　ISIS 电路设计与仿真：第一个电路设计

下面我们设计一个最简单的电路，并进行仿真运行。这个电路有三个元件：灯泡、开关和电池。首先，我们把元件选择到元器件列表窗口，元器件列表见表 1-1。

表 1-1　元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	LAMP	灯泡	12V
2	SWITCH	开关	
3	BATTERY	电池	12V

我们逐项选择上述元件到元器件列表窗口中，第一项是灯泡“LAMP”，在关键字文本框中输入“LAMP”，在结果窗口中我们选择“LAMP ACTIVE Animated Light Bulb”，用鼠标左键双击该项将它选到元器件列表窗口中，如图 1-19 所示。

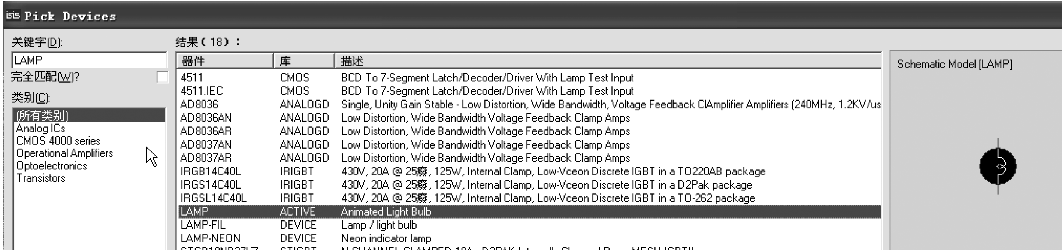


图 1-19　选择灯泡元件

不用关闭元器件选择对话框，继续其他两个元器件的选择。在关键字框中输入“SWITCH”，然后选择“SWITCH ACTIVE”项，用鼠标左键双击该项将它选到元器件列表

窗口中，如图 1-20 所示。

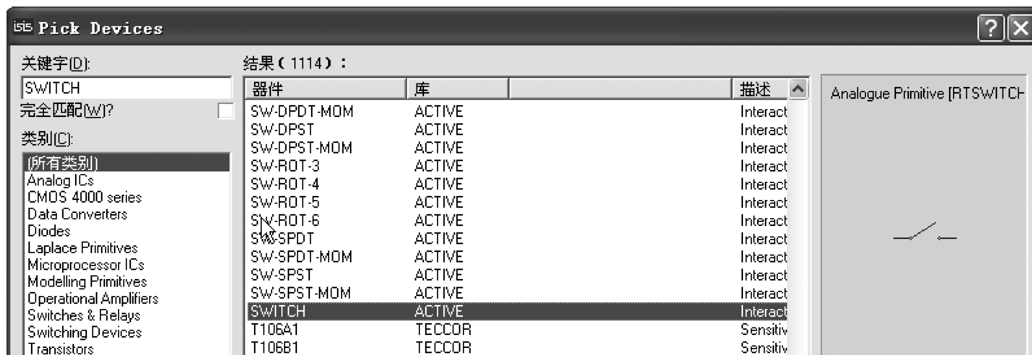


图 1-20　选择开关元件

在关键字框中输入“BATTERY”，在结果中选择“BATTERY ACTIVE DC Voltage Source,”用鼠标左键双击该项将它选到元器件列表窗口中，如图 1-21 所示。

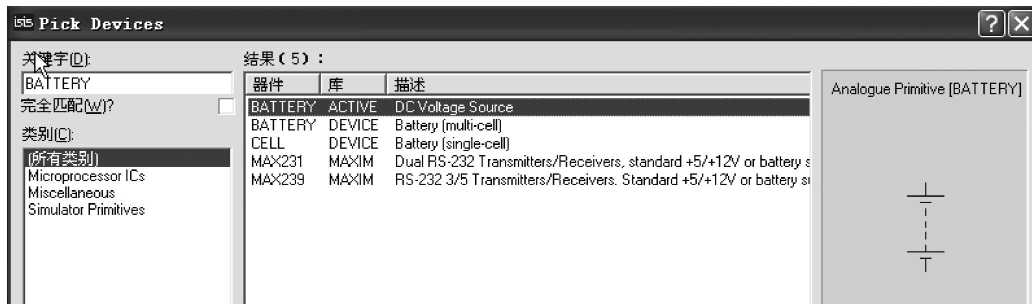


图 1-21　选择电池元件

添加选好元件后，在元器件列表窗口中就有了我们所需要的元件了，如图 1-22 所示。

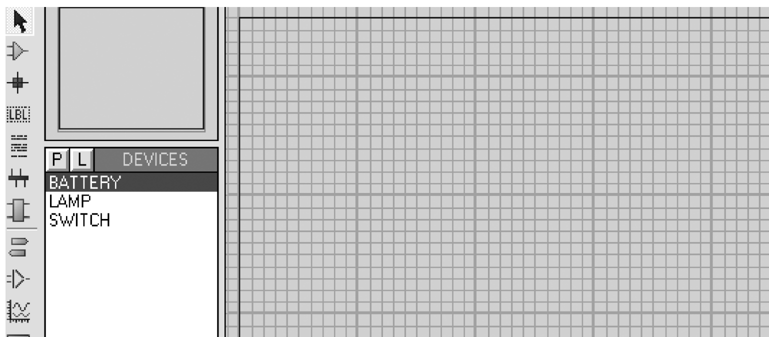


图 1-22　显示元件列表

现在，我们开始用这些元件搭建一个电路。首先我们要把这 3 个元件放到原理图编辑窗口的适当位置。在选用元件列表窗口中元件时，需要点击“模型选择工具栏”中的“元件模式”，以便切换到元件选择模式，显示所选的元件，如图 1-23 所示。

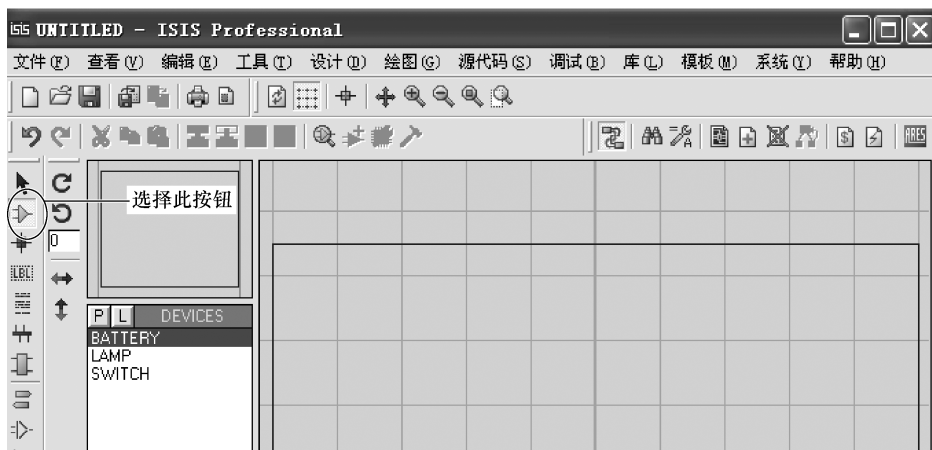


图 1-23　选择元件方式

选中一个元件，并用元件方向工具栏的工具调整元件的方向，然后将它放入原理图编辑窗口。例如，选择电池元件“BATTERY”，观察预览窗口，使用方向工具栏的旋转按钮使它正极朝上，如图 1-24 所示。

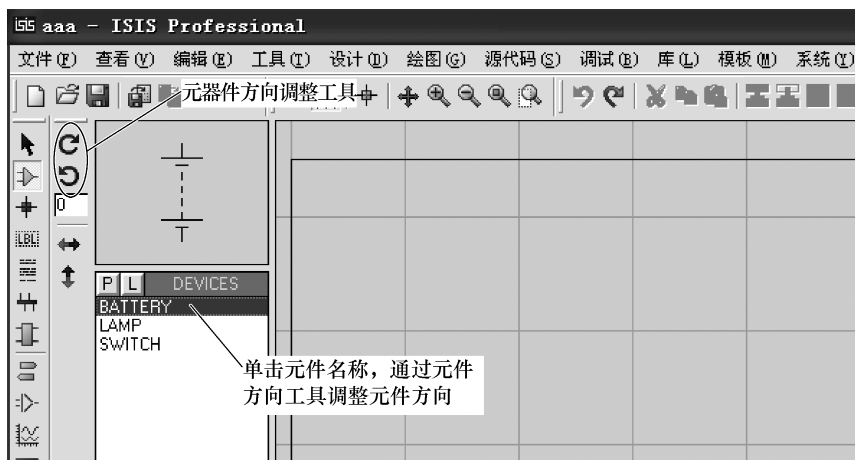


图 1-24　选择元件

然后将光标移动到原理图编辑区的位置，光标会变成铅笔状态，用鼠标左键单击原理图区域，出现元件图形，如图 1-25 所示。

该图形会随着光标移动，在你想放置的位置单击鼠标左键，元件就画在原理图上了，如图 1-26 所示，并且标上了标号和相关说明。当然，以后仍然可以移动位置。

通过以上方法，将 3 个元件安装要求摆放位置如下，如图 1-27 所示。

接下来，我们就要进行连线操作了。连线也非常方便，用鼠标左键单击某个元件连接的头部，拉向另一个元件的连接点，再次用鼠标左键单击即可。我们首先将电池和开关连接起来，如图 1-28 所示。

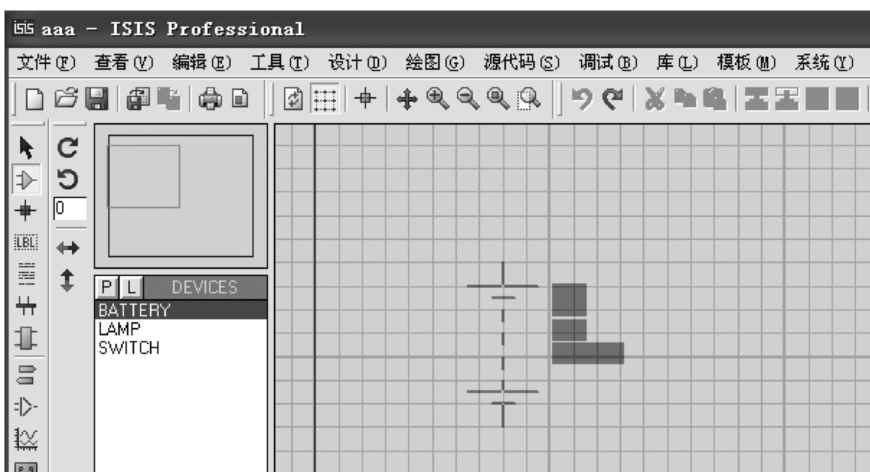


图 1-25　寻找放置元件位置

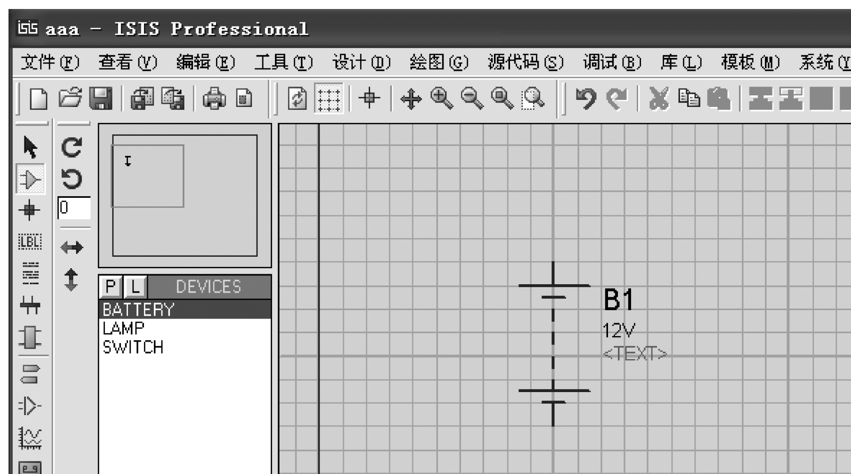


图 1-26　放置电池元件

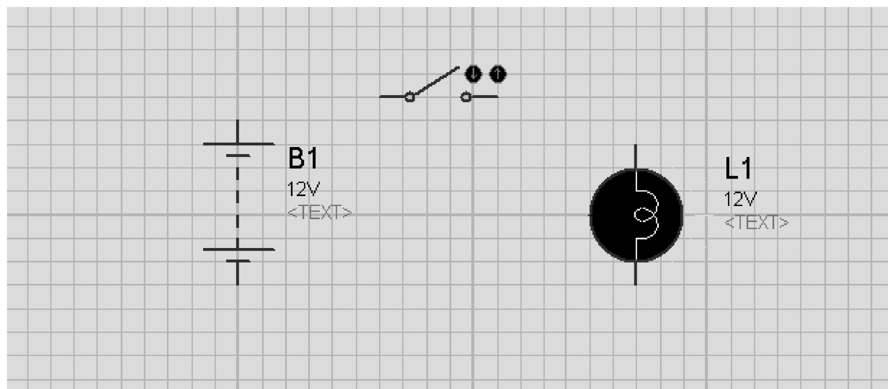


图 1-27　放置元器件

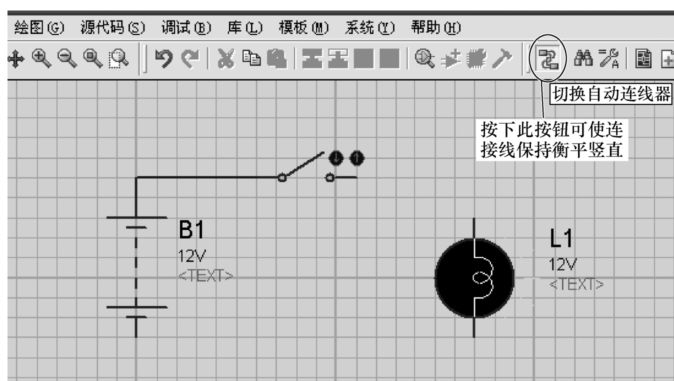


图 1-28　设置自动连线器

用同样方法，连接其他线路，完成电路设计，如图 1-29 所示。

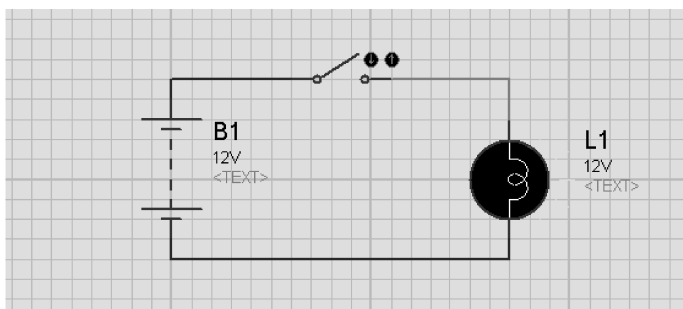


图 1-29　完成电路设计

为了使仿真效果更好，先要设置一下仿真的某些参数。依次打开菜单“系统”→“设置动画选项”，如图 1-30 所示。

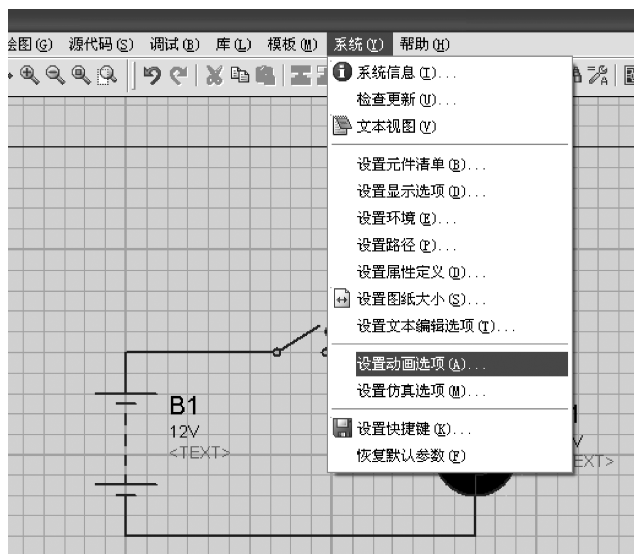


图 1-30　设置动画选项菜单

在弹出的对话框中，勾选“Show Wire Voltage by Colour?”和“Show Wire Current with Arrows?”（用颜色显示电路电压和用箭头显示电路电流方向），用鼠标左键单击“OK”键确认，如图1-31所示。

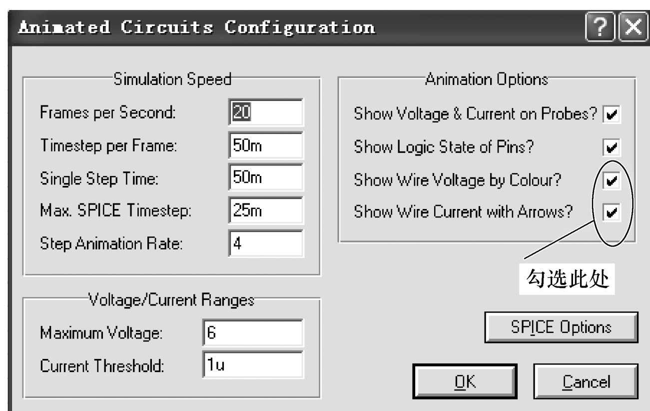


图1-31 设置动画选项

万事俱备，接下来就可以进行仿真了。将光标放在电路图的中间位置，通过滚动鼠标滚轮来调节使得设计图大小适中，或在预览图中调节图的大小和位置也行。然后，按下仿真启动按钮，开始仿真，如图1-32所示。

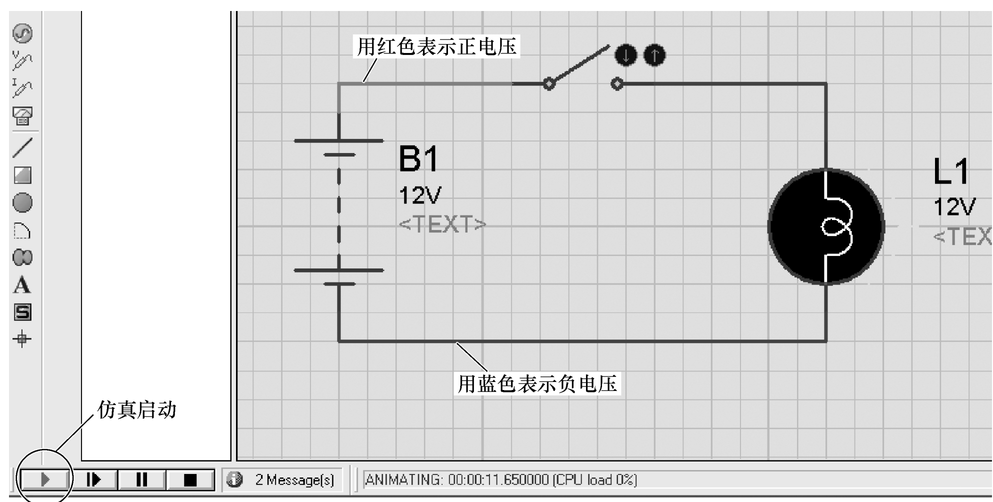


图1-32 启动仿真

现在，电路正负电压已经用颜色表示出来了，但是开关未合上，灯泡未点亮。接着，我们可以合上开关，观察电路的变化。用鼠标左键单击开关，开关就会合上，如图1-33所示。

开关合上了，电流方向用箭头表示出来，灯泡也亮了，并发出光芒。恭喜你取得了初步成功！

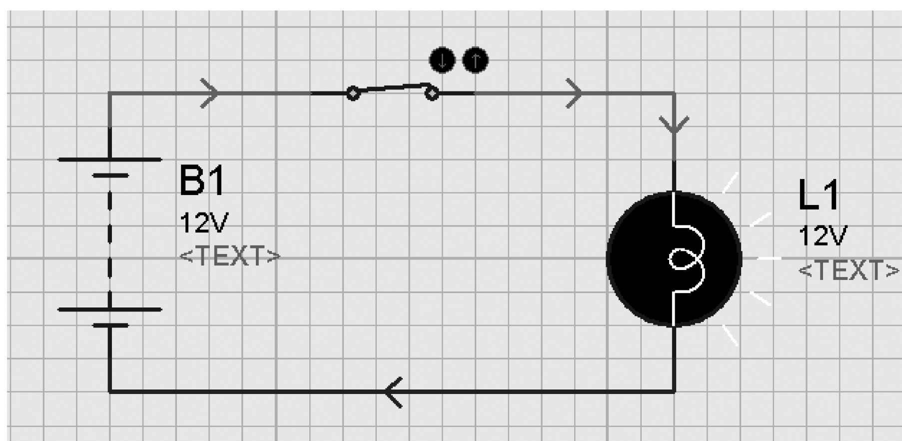


图 1-33　电源、开关、灯泡电路仿真

第2讲 元器件属性及编号设置

 导读

这一讲我们着重讲解如何设置元器件的属性和编号设置问题。元器件随着种类的不同会有不同的属性，如电压值、电阻值、封装等。元器件的编号不能相同，故需要按一定规律排列，如何做到方便、快速地设置编号也是一种技巧。

2.1 元器件的摆放

现在，我们来设计一个发光二极管应用的电路，所用到的元器件很简单，只有发光二极管、电阻和电源。这里不涉及变压部分，直接用5V电源。也没有用到单片机，所以不用编程和编译。但是有个新的问题，因为发光二极管有电流的要求，太小不发光，太大会损坏器件。这就需要根据欧姆定律计算电阻的值，并调整阻值。一般发光二极管的工作电流为10~20mA，外加电压为5V，设工作电流为20mA，则电阻值应为

$$\frac{5V}{0.02A}=250\Omega$$

设工作电流为10mA，则电阻值应为

$$\frac{5V}{0.01A}=500\Omega$$

计算结果，可选电阻值为200~500Ω。

下面我们开始设计电路，选择元器件见表2-1。

表2-1 元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	LED-BLUE	发光二极管	蓝色
2	LED-RED	发光二极管	红色
3	RES	电阻	标准，阻值可设定

选好元器件后，我们将发光二极管放置到原理图编辑窗口，这时需要调整元器件方向，可以用元器件方向工具来调整，但我们仍觉得不方便。现在介绍一种快捷键方法，将元器件向右旋转工具的快捷键改为空格键，步骤如下：

1）打开“系统”菜单，选择“设置快捷键”，如图 2-1 所示。



图 2-1　设置快捷键菜单

2）在编辑快捷键的对话框“命令组”中选择“方向工具条命令”，然后选择“顺时针旋转”工具，将光标点击在“快捷键”文本框，按键盘的空格键，文本框中即出现“Space”，按“确定”键结束，如图 2-2 所示。



图 2-2　设置元器件旋转快捷键

这样，就把“顺时针旋转”的快捷键变成空格键了，下面我们就来使用它。选择蓝色发光二极管 LED-BLUE，点击原理图编辑区，出现发光二极管形状，这时，我们按空格键，就会使得器件顺时针旋转，按一次，旋转 90°，如图 2-3 所示。

再一次单击鼠标左键，就会把发光二极管固定在原理图上，是不是很方便？

那么，如何调整已经放好位置的元器件呢？首先，用鼠标左键单击一下需要移动的元器件，将光标放在元器件位置，元器件四周就会出现虚线框，光标变成手状和一个十字，这时按住鼠标左键，就可以拖动元器件了。拖动时元件形状跟着鼠标走，在需要放下的位置释放鼠标左键，元器件就移动到新的位置了，如图 2-4 所示。也可以用这个方法移动元器件的其他内容，如编号、说明等。

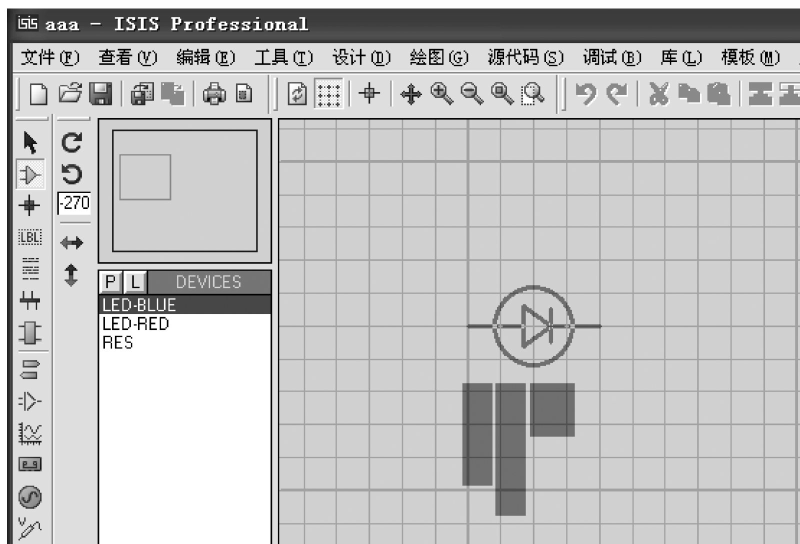


图 2-3　放置元器件

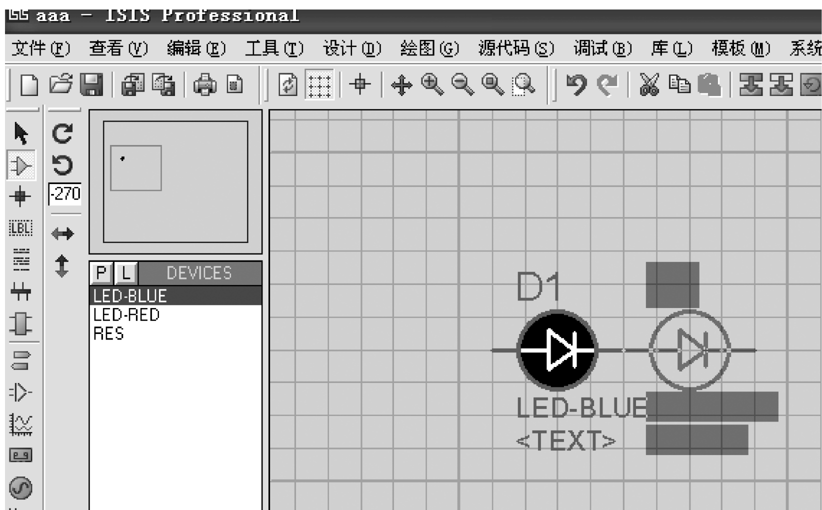


图 2-4　移动元器件

用上述方法，放置 2 个蓝色发光二极管、2 个红色发光二极管和 4 个电阻，我们发现，每个元器件除了编号、元器件名称或阻值外，还有一个说明 <TEXT>，会占用一定的空间（见图 2-5）。为了简洁起见，我们可以通过设置将它去掉。选择菜单“模板”→“设置设计默认值”（见图 2-6），打开对话框，在“隐藏对象”框中的“显示隐藏文本?”栏取消勾选，如图 2-7 所示。

这样，每个元器件的说明就去掉了，原理图变得比较简洁和紧凑。

下面，我们还要加上电源和接地标志。在哪里选择电源和接地标志呢？首先要在模型选择工具栏选择“终端模式”，这时，元器件列表窗口就会列出终端模式的元器件选项，其中的“POWER”就是电源，“GROUND”就是接地，如图 2-8 所示。

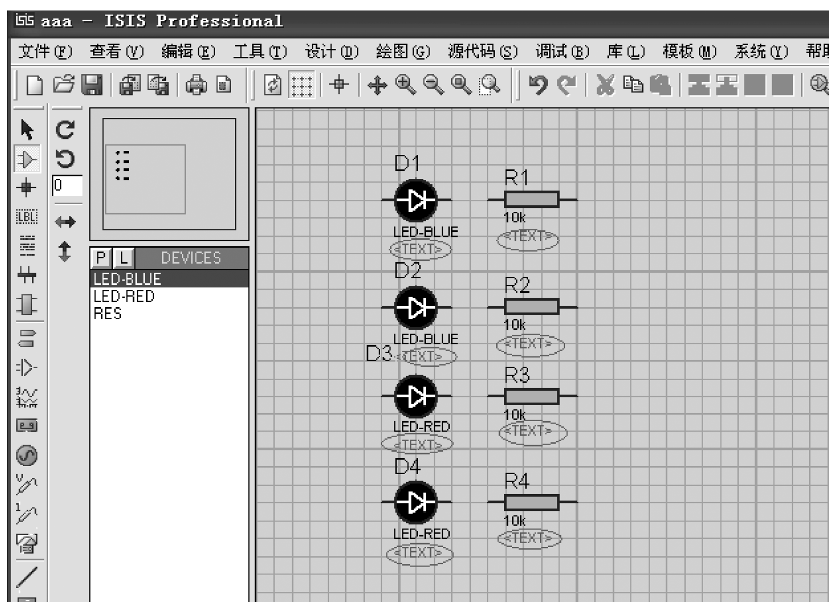


图 2-5　元器件说明值

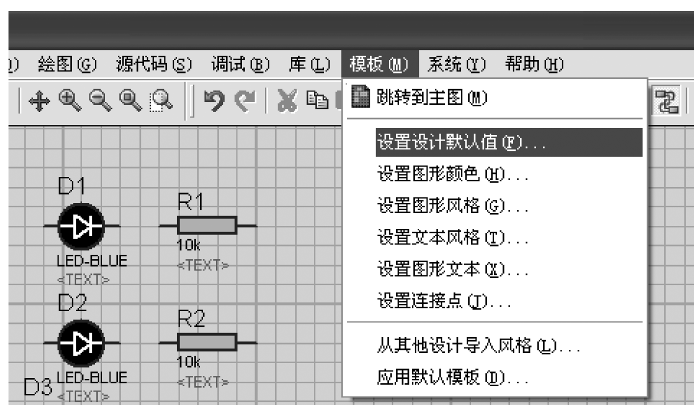


图 2-6　设置默认值菜单

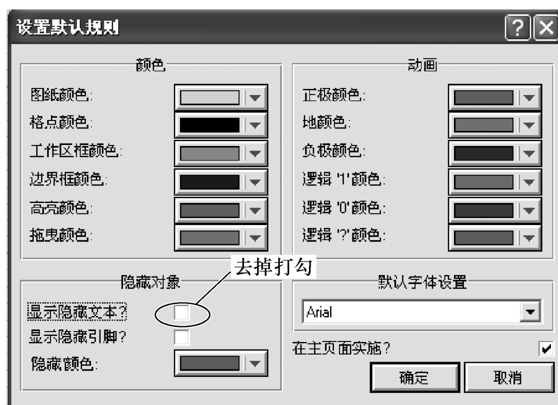


图 2-7　去掉元器件说明设置

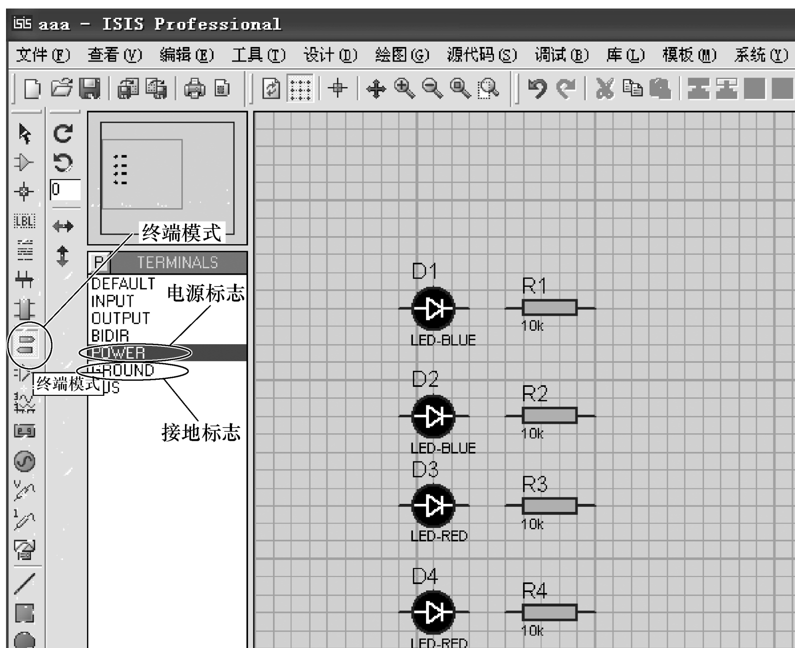


图 2-8　电源和接地标志

将电源和接地放置到电路中，并连接线路，如图 2-9 所示。

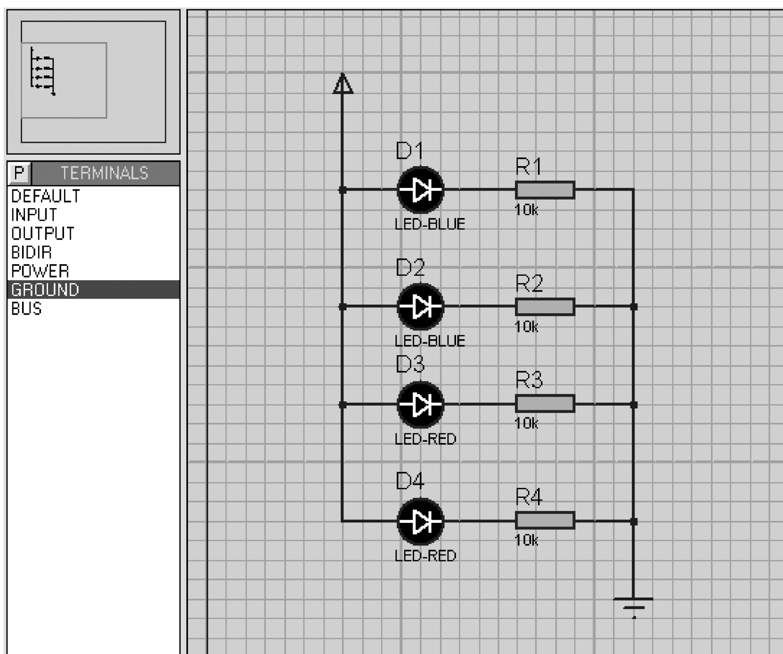


图 2-9　发光二极管电路

2.2 电路连线的编辑

在第 1 讲中我们进行过电路的连接，并使用了“切换自动连接线”按钮，使得连接线路能横平竖直。在连接线路时，如何使线路按照指定的路线走呢？我们可以在指定点设置拐点，让线路沿着拐点走，如图 2-10 所示，在 D5 和 R5 的连线中设置的 4 个拐点，线路就会按照拐点的设置走线。拐点的设置就是在连线时用鼠标左键单击指定点，形成拐点。为了画斜线，只要把“自动连接线”按钮设置成弹起状态即可，如图 2-11 所示。

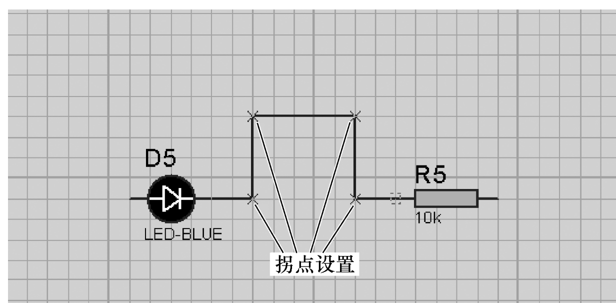


图 2-10　设置拐点

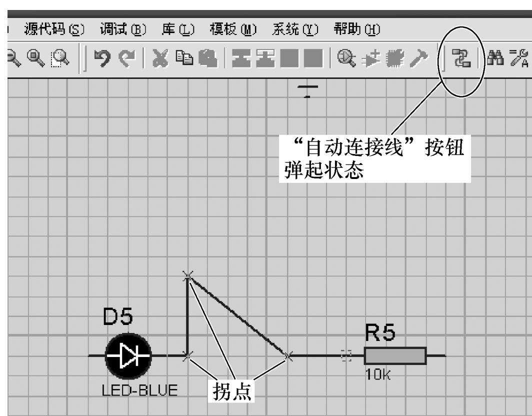


图 2-11　画连接线斜线

那么，如果线路画错了，如何删除线路呢？可以有几种方法，一种是用鼠标右键单击需要删除的线路，在弹出菜单中选择“删除连线”，如图 2-12 所示。

另一种更快捷的方法是，用鼠标指向需要删除的线路，然后用鼠标右键双击线路即可删除。

对于画好的线路，也可以改变它的走向，用鼠标指向需改变的线路，点击鼠标右键，在弹出菜单中选择“拖拽连线”，就可以用鼠标拖动线路了，如图 2-13 所示。或者直接用鼠标左键按住线路拖动，在拖动时，若“切换自动连接线”按钮按下，就只能横平竖直地拖动，没有按下就可以按任意方向拖动。

若要删除元器件，只要选中需要删除的元器件，用鼠标左键单击某个元器件，或用鼠标拉出一个框围住若干个元器件，然后敲击键盘“Delete”键或用鼠标右键单击，在弹出的菜单中选择“剪切”即可。



图 2-12　删除连线

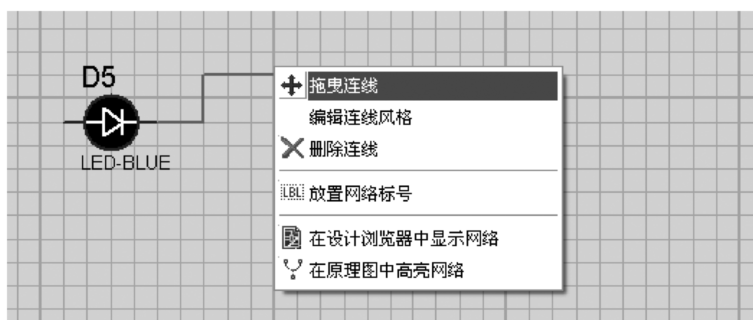


图 2-13　拖动元器件

2.3　设置电源属性

在 2.2 节中，我们已经画好了电源标志，如何设置它的属性呢？编辑每一个元器件的属性，只要双击该元器件，就会打开属性编辑对话框。现在我们来编辑电源的属性，用鼠标左键双击电源标志，打开电源属性对话框，如图 2-14 所示。

在 Label 页面中的标号下拉菜单中，选择 VCC 或 VCC/VDD。这时默认设置的方法，电源值为 +5V，你也可以直接设置成 +5V。有时某些元器件需要特殊的电源值，如 +15V、−15V等，就可以直接设置，但前面一定要加上正负号，如图 2-15 所示。

同样，接地的属性可以选择 GND，也可以不设置。



图 2-14　电源属性对话框

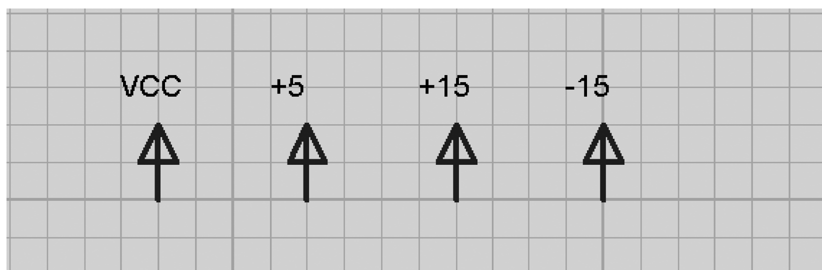


图 2-15 设置电源

2.4 设置元器件属性

对于每一个元器件，有 3 个属性是必须设置的，一个是元器件参考，就是编号，默认是自动编号，编号应该是惟一的。另一个是参数值，如电阻就是电阻的阻值。还有一个就是元器件的封装，用于制作电路板。拿电阻来说，用鼠标左键双击电阻元件，打开属性对话框，我们就可以修改该元件的属性了，如图 2-16 所示。

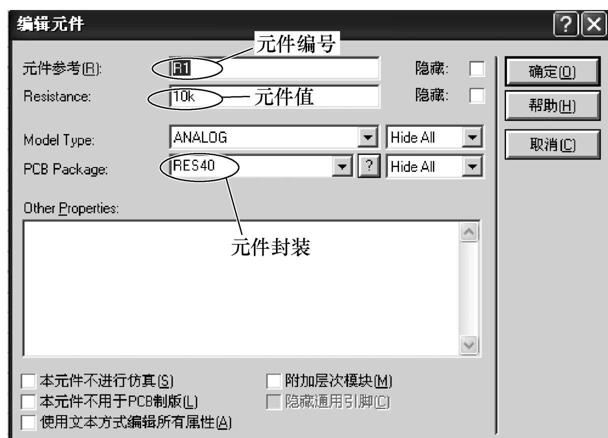


图 2-16 电阻属性

每一种元器件不同，属性对话框也会不同，用鼠标左键双击发光二极管，打开属性对话框，如图 2-17 所示。

元器件的封装在原理图设计时可以不设置，不影响仿真。但在制作电路板之前必须设置完毕，否则不能进入 PCB 制作环节。封装是否设置完成，可以通过“设计浏览器”来检查，如图 2-18 所示。

以上我们已经知道如何设置元器件的属性了，现在就可以把电阻的值改为我们希望的值了。标准电阻的默认值为 10kΩ，在未改动前我们可以仿真一下，看发光二极管是否会亮，如图 2-19 所示。

运行仿真后，发光二极管并没有亮。发光二极管会亮有两个条件：一是电源极性必须正确；二是电流大小必须合适，而电流大小是由电阻值决定的。由于电阻值太大，电流就会很



图 2-17　发光二极管属性

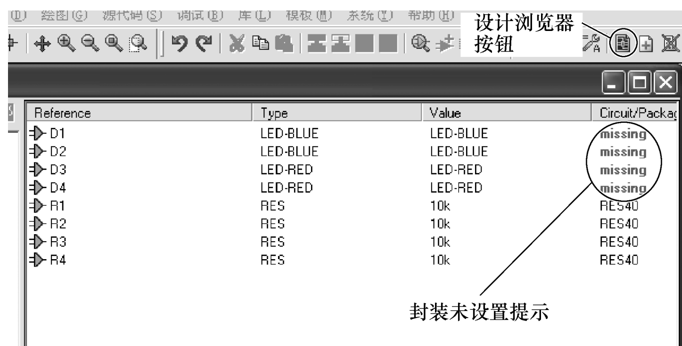


图 2-18　查看元器件封装

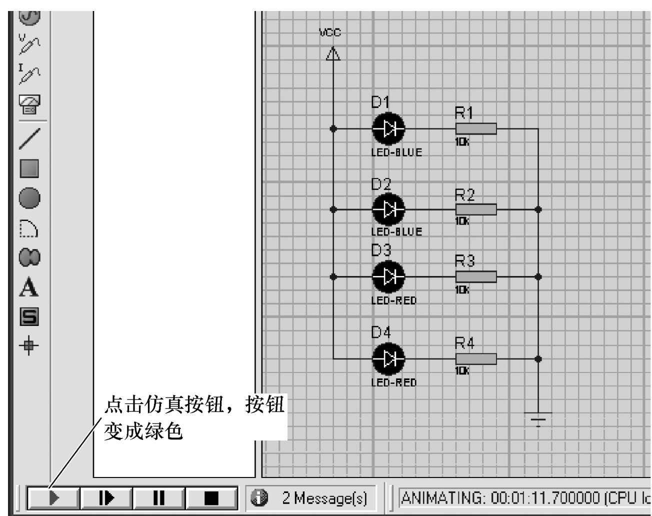


图 2-19　运行电路仿真

小，虽然极性正确，也不能发光。现在我们把每个电阻值改为 500Ω 再试一次，如图 2-20 所示。要注意，修改属性时，必须停止仿真运行！

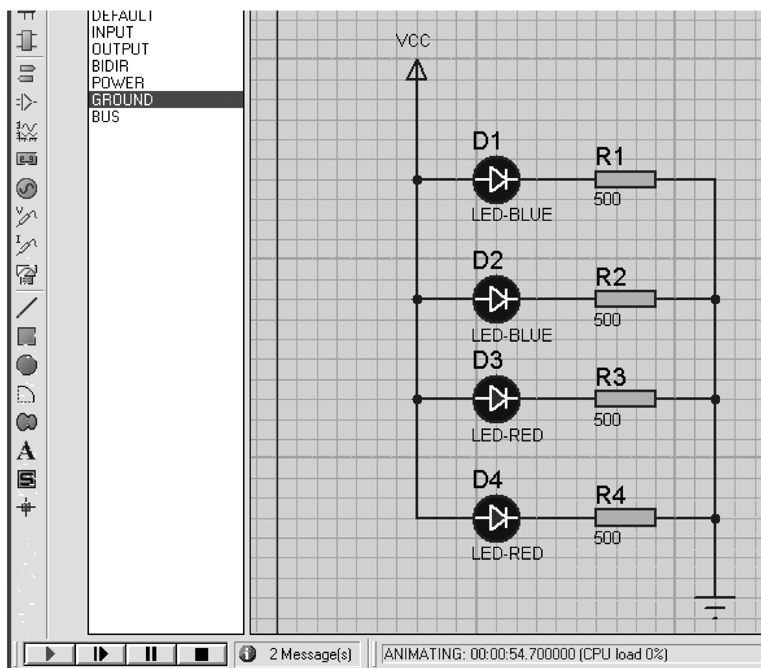


图 2-20　发光二极管点亮

现在 4 个发光二极管都亮了，恭喜你获得成功！

2.5　用工具快速设置元器件属性

在以上属性设置中我们发现，一个一个地设置很慢，也不太方便。有没有更方便的方法呢？有！我们需要借助“属性设置工具”来做。依次打开菜单“工具”→“属性设置工具”，如图 2-21 所示。

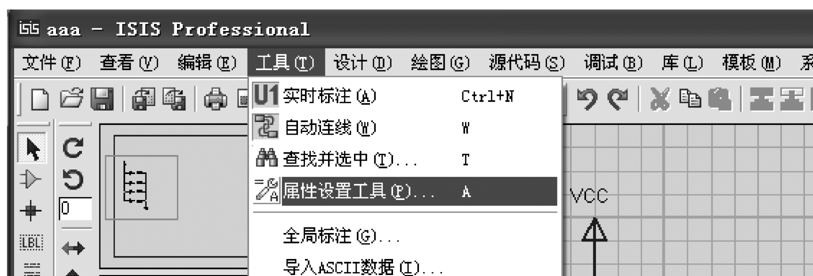


图 2-21　使用属性设置工具

有几种方法可以实现快速修改属性，根据参数的不同和选项的不同，可以灵活设置各种属性。首先，我们用点击的方法实现属性的修改。因为我们需要将所有的电阻值都改为 500Ω，打开“属性设置工具”后，在“字符串”文本编辑框中输入 VALUE = 500，并在

“应用到”选择“点击”，如图 2-22 所示。

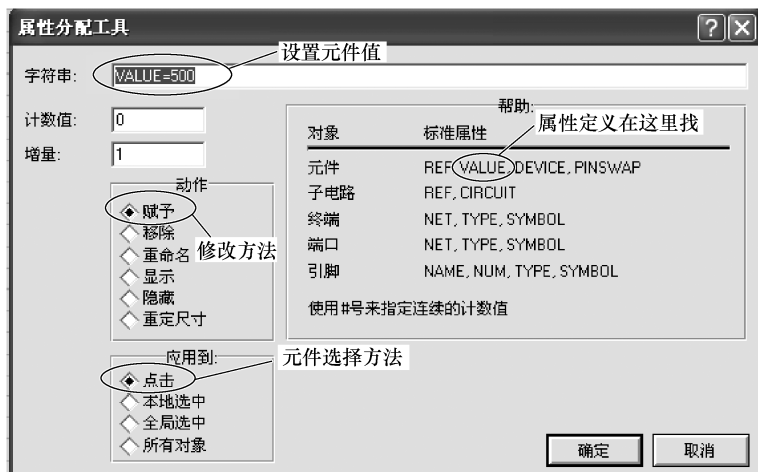


图 2-22　用属性设置工具设置电阻值

用鼠标左键单击“确定”按钮后，分别用鼠标点击每个需要修改的电阻，就会把电阻值改为 500Ω，如图 2-23 所示。

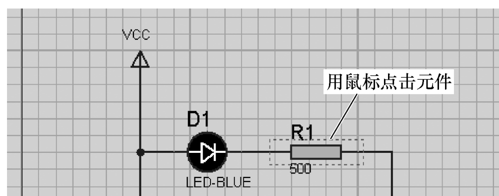


图 2-23　修改元件属性

另一种方法是，首先选中需要修改的对象，如图 2-24 所示。

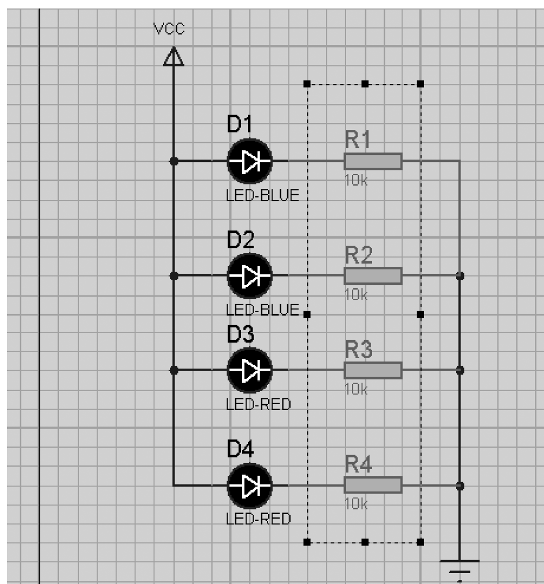


图 2-24　选中需设置属性的元件

然后打开“属性设置工具”，设置需要的阻值，并选择“本地选中”或“全局选中”，如图 2-25 所示。

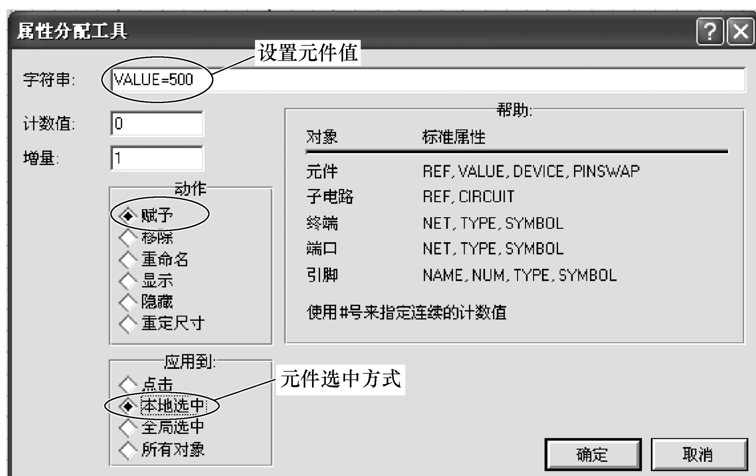


图 2-25　用属性设置工具设置元件值

用鼠标左键单击“确定”按钮，就会将选中的电阻值都改为 500Ω。用这个工具也可以修改元件的编号，即元件参考值，在“字符串”中需要用 REF 来定义，并且需要按顺序增量编号，用#号代表编号，在“计数值”中设置起始值，“增量”中设置增加的数值。例如，我们需要把电阻的编号改为 R10 ~ R13，可以用点击修改或全局修改。现在我们用全局修改方式，选中所有的电阻，然后设置“属性设置工具”，如图 2-26 所示。

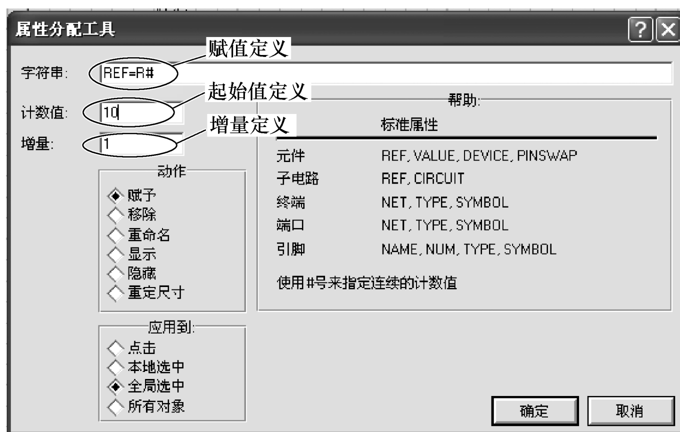


图 2-26　用属性设置工具修改电阻编号

用鼠标左键单击“确定”按钮后，就把选中的电阻编号修改好了，如图 2-27 所示。

也可以采用点击的方式，点击一个修改一个，从 R10 开始，依次增加编号。

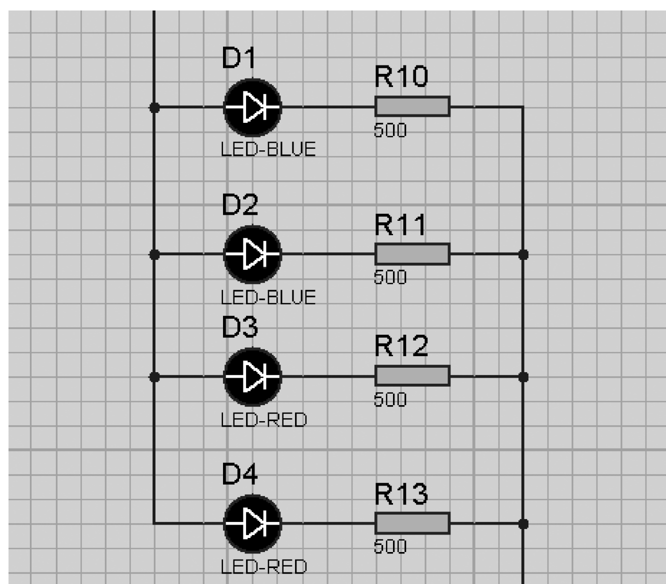

D1
R10
500
LED-BLUE
D2
R11
500
LED-BLUE
D3
R12
500
LED-RED
D4
R13
500
LED-RED

图 2-27　修改电阻编号

第3讲

单片机程序编写和编译

导读

前面2讲我们把 Proteus ISIS 的基本用法介绍了一下。从本讲开始，我们就要使用单片机来设计电路应用了。首先，要介绍一下 Keil C51 的使用方法，因为要用它来编写和编译程序，供 Proteus 仿真运行。然后我们设计一个最简单的单片机应用，用单片机点亮一个发光二极管，并用 Keil C51 编写和编译的程序仿真运行。

3.1 Keil C51 使用方法

这里我们使用 Keil C51 uVision2 软件，虽然更高的版本可以与 Proteus 联机调试，但需要安装插件，会增加一些困难。因此，选用较低的版本，更有利于简化学习过程。

首先，我们介绍一下用 Keil C51 编写和编译 C 语言程序的基本步骤。点击 Keil uVision2 软件的菜单或图标，进入 Keil C51 界面，在菜单栏选择“工程”→“新建工程”，如图 3-1 所示。

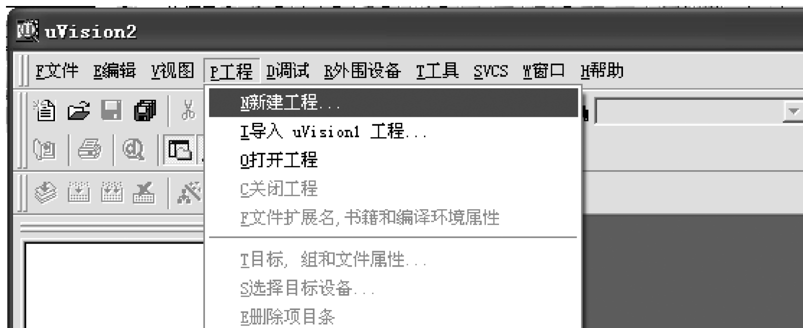


图 3-1　新建工程菜单

然后在弹出的对话框中选择一个路径，新建一个目录，用以存放工程文件，并输入一个工程名称，例如，在“我的文档”中新建一个目录 test，如图 3-2 所示。

然后进入新建的目录 test，并将工程命名为 t1，用鼠标左键单击“保存”按钮退出，如图 3-3 所示。

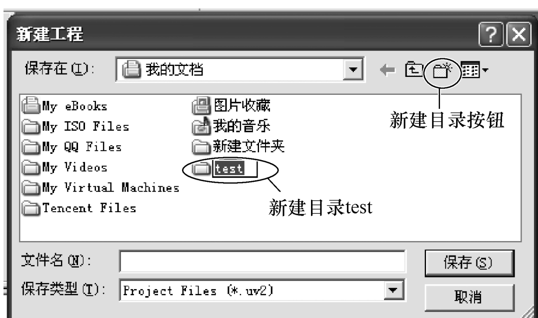


图 3-2　新建工程目录



图 3-3　新建工程名称

接着，要选择单片机型号，从厂商开始选，找到一个具体的型号。我们采用 Atmel 公司的 89C52 单片机，如图 3-4 所示。

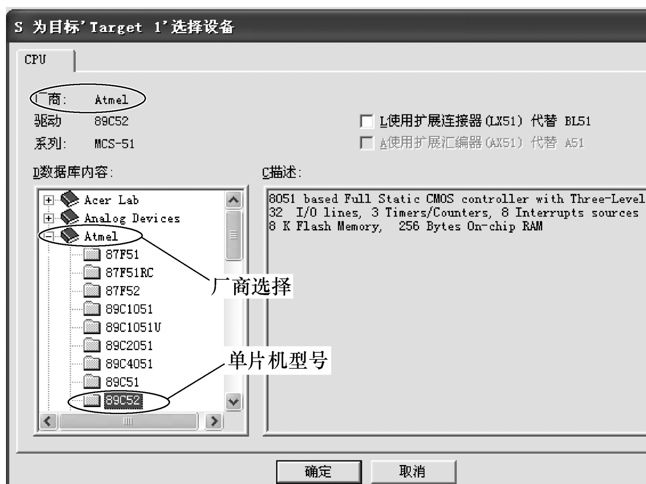


图 3-4　选择厂商和单片机型号

用鼠标左键单击“确定”按钮结束。这样，一个工程就建好了。接着，要向工程中添加文件。一般是新建一个文本文件，然后通过命令添加至工程的文件夹中，如图 3-5 所示。

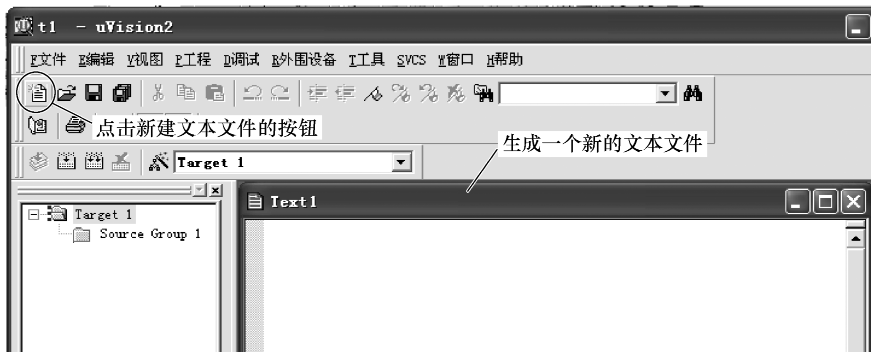


图 3-5　新建程序文件

我们新建一个文本文件，并在其中输入 C 语言程序，程序内容如下：

```
#include <reg51.h>
sbit LED = P0^0;
void main()
{
    while(1)
    {
        LED = 0;
    }
}
```

输入程序内容后，界面显示如图 3-6 所示。

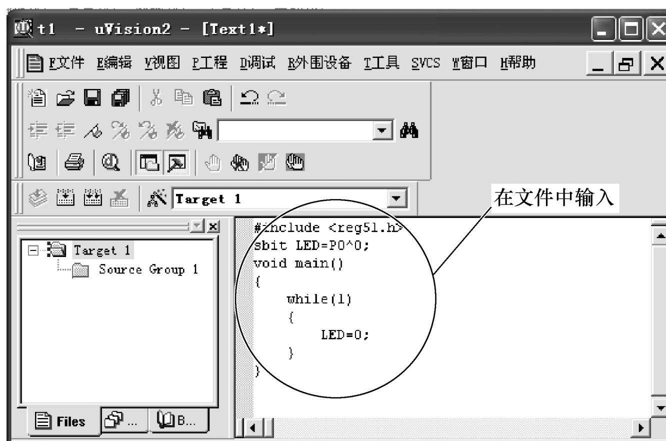


图 3-6　输入或粘贴程序

然后，将该文件存盘保存，如图 3-7 所示。

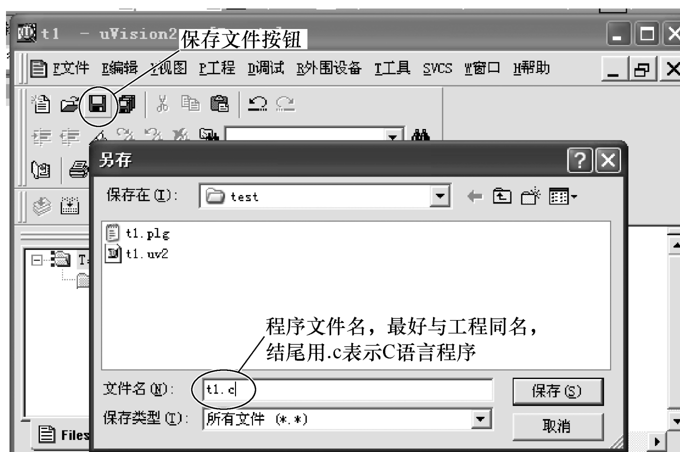


图 3-7　保存 C 语言程序文件

用鼠标左键单击“保存”按钮退出结束。虽然文件已经存放在工程文件夹中，但并没有加入到工程中，还需要通过命令，将文件加入工程，操作过程如图 3-8 所示。

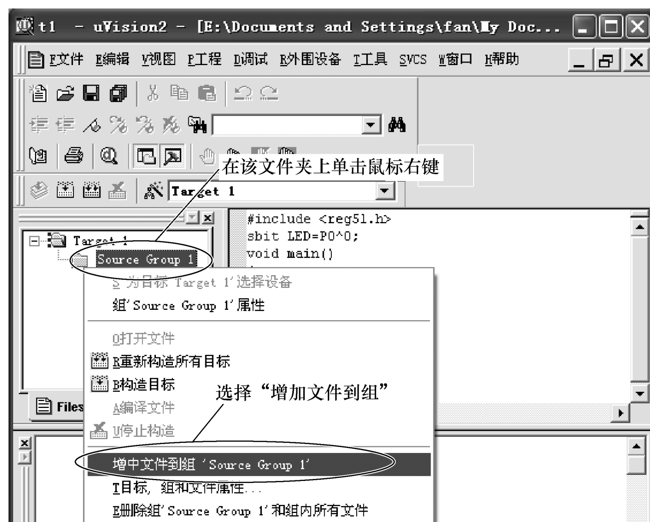


图 3-8　选择添加文件的菜单

在弹出的对话框中选择保存的 C 语言文件，如图 3-9 所示。

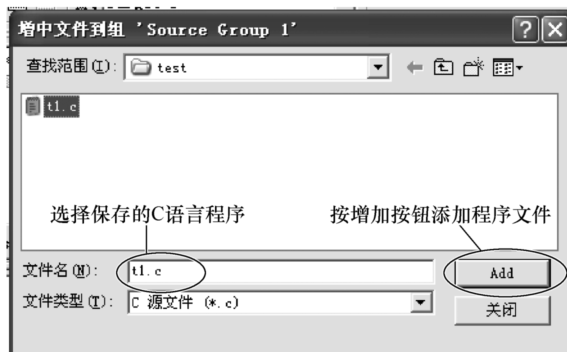


图 3-9　选择、添加程序文件到工程中

按“Add”按钮添加文件，若有多个文件可多次选择添加，最后按“关闭”按钮结束。添加文件后，我们就可以在该目录下看到添加的文件了，如图 3-10 所示。

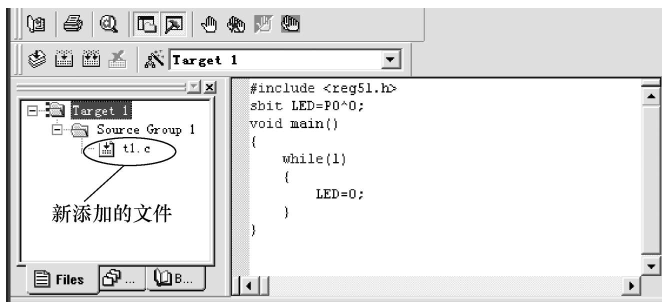


图 3-10　查看已添加的程序文件

接下来，需要编译该工程，产生可执行文件供 Proteus 仿真运行使用。在编译前，先要设置目标属性。将光标放在最上层等目录“Target 1”，依次打开菜单“工程”→“目标 Target 1 属性”，如图 3-11 所示。

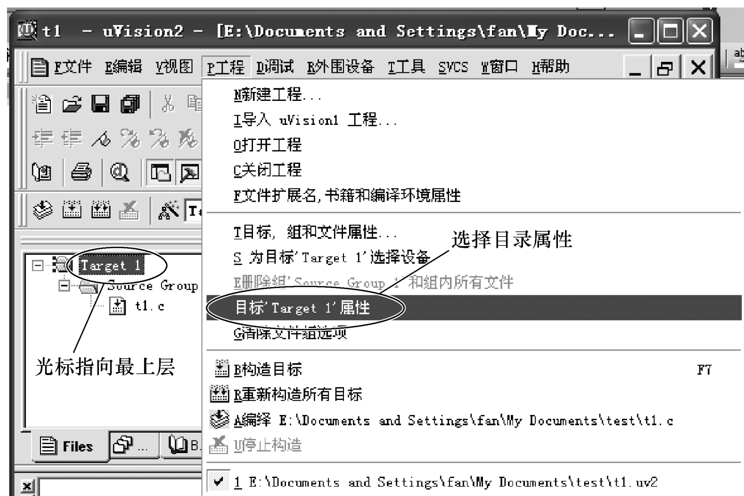


图 3-11　设置目标属性

然后在对话框中选择“输出”页进行设置，选择“生成 HEX 文件”即可，如图 3-12 所示。

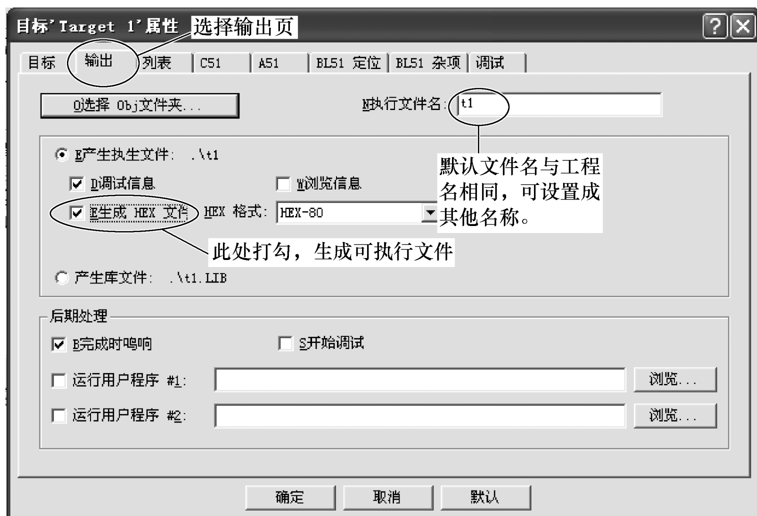


图 3-12　设置生成可执行文件

用鼠标左键单击“确定”按钮保存设置。设置完目标属性后，就可以进行编译了。点击菜单“工程”→“构造目标”，如图 3-13 所示。

系统开始编译工程，并在编译输出框中显示编译结果，如图 3-14 所示。

从显示结果看，在编译中没有错误、没有警告，并产生了 HEX 文件即可执行文件，编

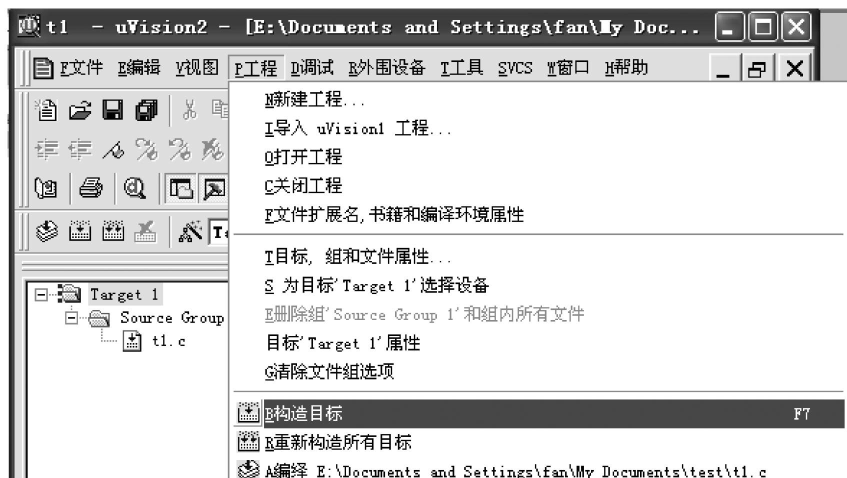


图 3-13　编译程序

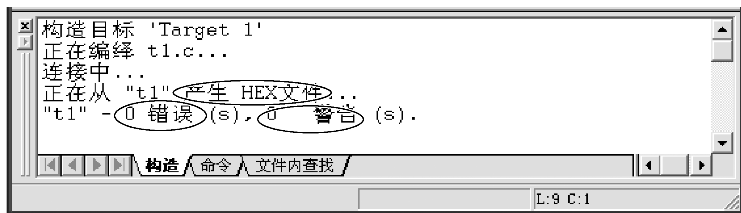


图 3-14　编译结果显示

译成功了！查看工程目录我们可以找到生成的可执行文件 t1. hex，如图 3-15 所示。

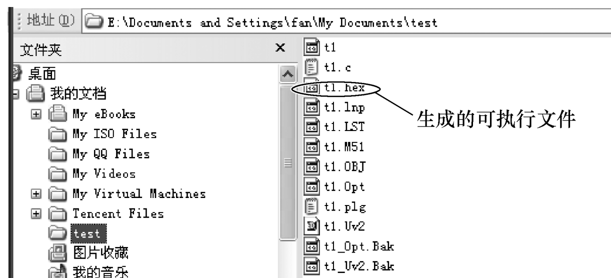


图 3-15　查看生成的可执行文件

到这里，我们已经完成了用 Keil C51 编写和编译 C 语言程序，生成了可执行文件。

3.2　第一个 C 语言程序：单片机点亮一个发光二极管

在上一节我们编写和编译的 C 语言程序功能，其实就是“单片机点亮一个发光二极管”，只是没有解释程序的每一句的作用而已。本节我们就来详细地讲一下如何用 C 语言设计一个单片机应用程序。

首先，还是列出已经编好的程序，并编上号，一边讲解：

```
#include <reg51.h>            //单片机头文件
sbit LED = P0^0;              //定义一个发光二极管
void main()                   //主程序
{
    while(1)                  //无限循环
    {
        LED = 0;              //发光二极管点亮命令
    }
}
```

有的读者担心没有系统地学过 C 语言编程，这是不对的。随着课程的深入，你会发现自己已经学会了 C 语言编程，并没有感到什么困难。C 语言是一种高级语言，很接近人类的语言或算术公式，所以，应该很容易理解和接受。有些语法和规则只要套用就可以了，没有必要深入研究和探讨。我们的目的只有一个，就是用 C 语言告诉单片机按照我们的要求工作。

先来看一下程序的基本框架，一般单片机应用的 C 语言程序的框架是：

```
1    单片机头文件          #include <reg51.h>
2    主程序                void main()
                           {
3    无线循环                  while(1)
                               {
4    工作内容(循环体)              ……
                               }
                           }
```

上述 1 ~3 项是每个程序都有的，你可以一开始就把它写好，然后再添加所需的内容。工作内容视具体的要求不同而异。

单片机头文件是引用一个定义了单片机应用所用到的变量、函数的文件。你要编写哪种单片机应用程序，就要引用那种单片机的头文件。我们现在要用 51 系列的单片机，就引用 reg51.h 或 reg52.h 文件，用#include 命令引用。

其次是主程序，每一个 C 语言程序都有一个主程序 main（），void 是没有返回值的意思。主程序下边有一对大括号｛｝，所有的程序内容写在里面。括号是成对出现的，编写的时候最好上下对齐，以便检查发现错误。

在主程序中，一般会有一个无限循环 while（1），1 是条件永远为真的意思。这是因为单片机程序是裸机跑的程序，没有操作系统，就这一个程序在不停地运行，所以只能让它无限循环了。它的下面也是一个大括号，包含循环内容，即单片机的工作内容。

剩下的就是工作内容，需要根据实际需要编写。这个程序只有一条语句，让发光二极管点亮。这就涉及硬件问题，单片机 C 语言编程与一般的 C 语言编程不同之处就是针对硬件设置来编写程序。单片机通过它的端口即它的引脚输出高、低电平，来控制外部设备工作的。

现在，它要控制发光二极管工作，它的一个引脚接在发光二极管的负极，发光二极管的正极通过电阻接电源正极，因此，当单片机引脚为低电平时，发光二极管就会导通点亮。反之，当单片机引脚为高电平时，发光二极管就不会导通而不亮。所以，只要控制单片机引脚的高低电平就能控制发光二极管的开启和关闭。

在程序中，我们首先定义连接发光二极管的引脚 P0 口的第一个引脚，用 P0^0 表示，这是在单片机头文件中定义的。89C52 单片机共有 4 个输入输出端口，分别是 P0、P1、P2 和 P3。每个端口有 8 各引脚，用 0 ~ 7 编号，所以 P0 口就有 P0^0 ~ P0^7 共 8 个引脚，其他端口也是一样。可以为每个引脚定义一个变量名，以便于使用。因为一个引脚就是一位，所以该变量的类型是“sbit”，定义如下：

```
sbit LED = P0^0;        //定义 LED 为端口 P0^0
```

在程序语句中用“;”结束，“//”后面是程序的注释内容，解析程序语句的作用。有了定义，我们只要给变量赋值，该端口就会按照赋值输出高低电平。赋值 1 输出高电平，赋值 0 输出低电平。

```
LED = 0;                //P0^0 输出低电平，发光二极管点亮
```

由于该语句在无限循环 while（1）中，发光二极管就会一直亮着。如果再加上一句：

```
LED = 1;                //P0^0 输出高电平，发光二极管关闭
```

它就会一亮一灭，即不断地闪烁，当然，在中间还要插入延时，适当延长间隔才能使眼睛能看出来。

3.3　设计电路和仿真运行程序

以上我们解释了程序的编写基本框架和针对硬件编写 C 语言程序的一般原理，接着，我们就要根据要求设计一个单片机点亮一个发光二极管的电路，运行我们的程序。打开 ISIS 设计环境，选择元器件见表 3-1。

表 3-1　选择元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	LED-BLUE	发光二极管	蓝色
2	RES	电阻	标准，阻值可设定
3	89C52	单片机	51 系列

这些元器件，除了单片机我们都熟悉。用同样的方法，把单片机元件放到元器件列表窗口中。在元器件列表窗口中按“P”键，在关键字中输入“89C52”即可找到该单片机元件。将发光二极管负极连接到 P0^0 引脚，电阻值改为 500Ω，接电源，如图 3-16 所示。

电路画好了，把程序装入单片机，就可以仿真运行了。双击单片机，弹出“编辑元件”对话框，在“Program file”文本框的对应栏，点击浏览按钮，查找编译好的可执行文件，如图 3-17 ~ 图 3-19 所示。

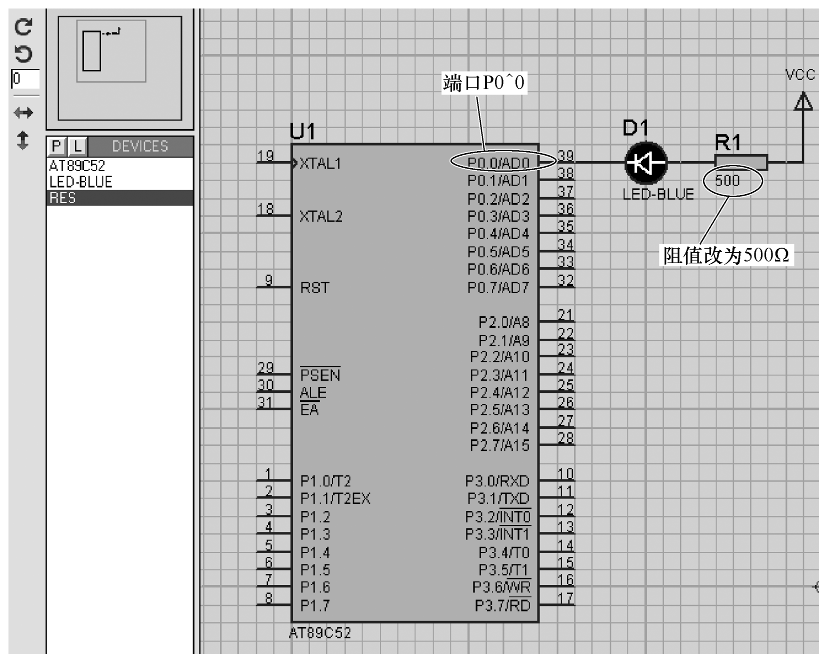


图 3-16　点亮一个发光二极管电路



图 3-17　查找按钮

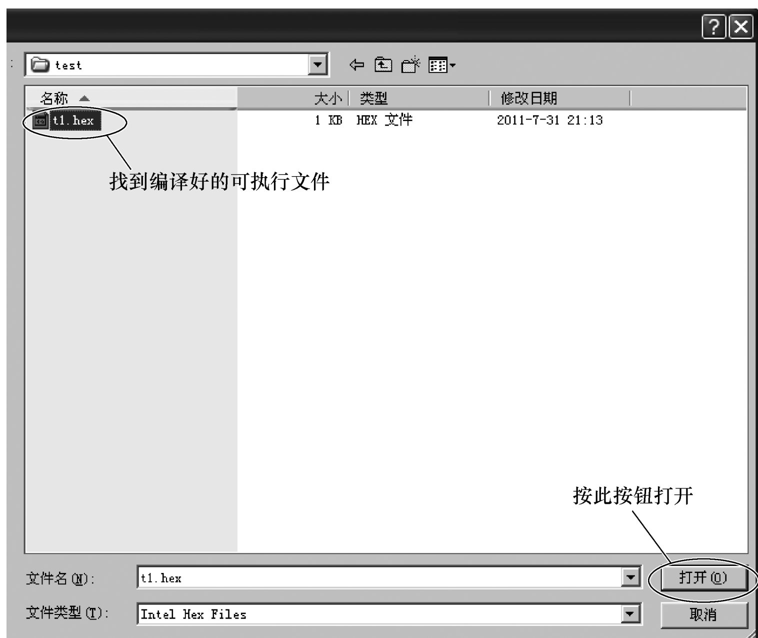


图 3-18　确定找到的可执行文件



图 3-19　查找可执行文件

这样，我们就好像在单片机中烧写了可执行程序。下面，就可以仿真运行了。用鼠标左键单击仿真按钮，开始仿真，如图 3-20 所示。

至此，我们实现了第一个单片机应用仿真，恭喜你！

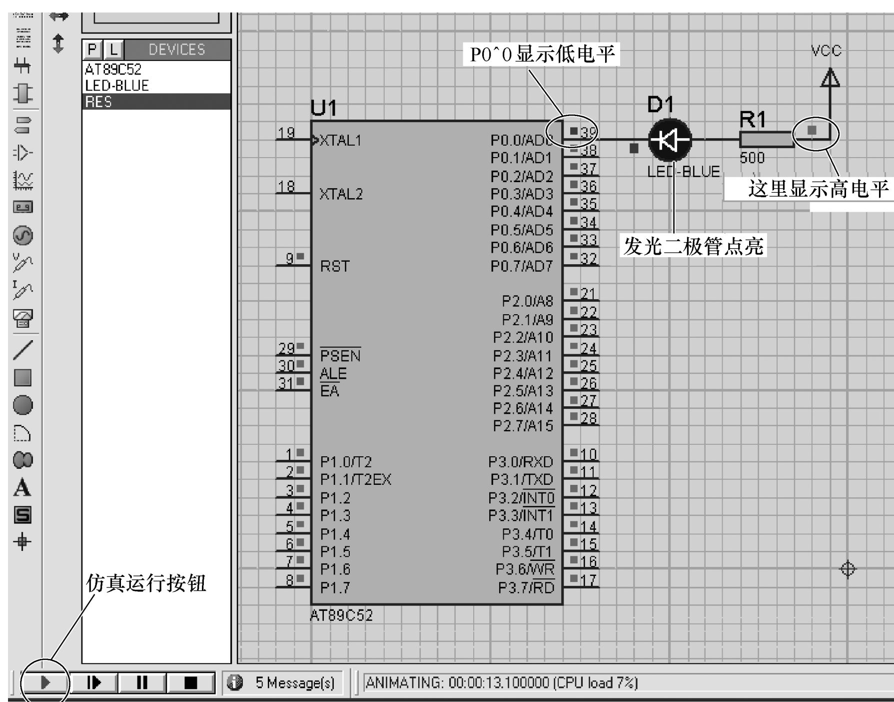


图 3-20　点亮一个发光二极管

第4讲

节点、总线连接技术

 导读

在这一讲里，我们要连接更多的元器件至单片机，由于数量多、距离远的问题，直接相连就会有困难，因此，要借助其他连接技术，节点和总线连接方法就是很好的解决方案。节点表示相同的网点，只用标记而不需要连线就表示连接在一起了，就像电源和接地一样。总线则是节点在局部的运用，也减少了连线数量，使电路表示更清晰。

4.1 单片机流水灯控制电路

这一节我们要设计一个流水灯控制电路，也叫跑马灯。用8个发光二极管连接至单片机，使发光二极管按照一定的方向、速度逐个点亮和关闭。以下我们分别用节点和总线的方法连接发光二极管。首先在原理图编辑区放置单片机、8个发光二极管和8个电阻，并采用第2讲介绍的方法，把元器件的说明去掉（在“模板”→“设置设计默认值”中的“显示隐藏文本”项的勾选去掉），把8个电阻值都改为500Ω（用“工具”→“属性设置工具”），在发光二极管正极连接电源。元器件列表见表4-1。

表4-1 元器件列表

序　号	元器件名称	名称说明	备　注
1	LED-BLUE	发光二极管	蓝色
2	RES	电阻	标准，阻值可设定
3	89C52	单片机	51系列

连接发光二极管和电阻，只要连接第一根线，以下用鼠标左键双击连接点即可完成连接，加快连线速度，如图4-1所示。

我们先试用节点连接的方法连接至单片机。在发光二极管和单片机P0口接上节点。在“终端模式”下，选择“DEFAULT”，选择了默认节点，如图4-2所示。

然后在发光二极管和单片机P0口放置节点，并连线，如图4-3所示。

接着，要给每个节点编号，以便连接到指定的点。节点的编号就是网络号，用NET表示，2个节点编号相同，就表示是连接在一起的。编号的标注仍然可以用“属性设置工具”来快速设置。把发光二极管的8个节点编号设置成与单片机的8个节点一一对应就行了。首

先选中发光二极管的 8 个节点，并打开“属性设置工具”，如图 4-4 和图 4-5 所示。

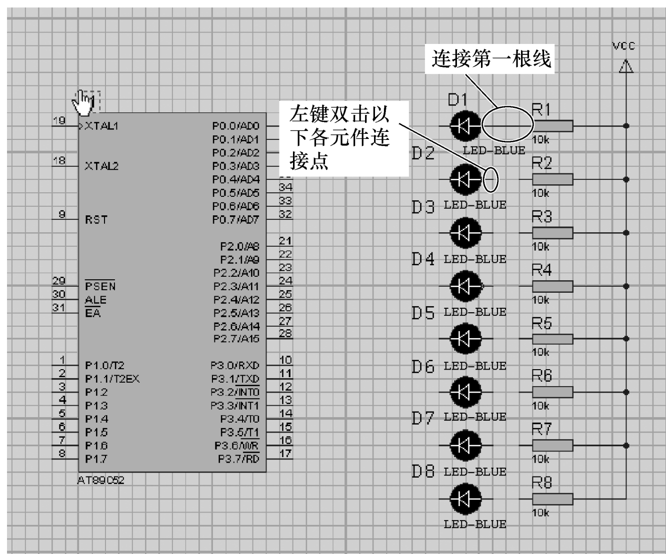


图 4-1　快速连线

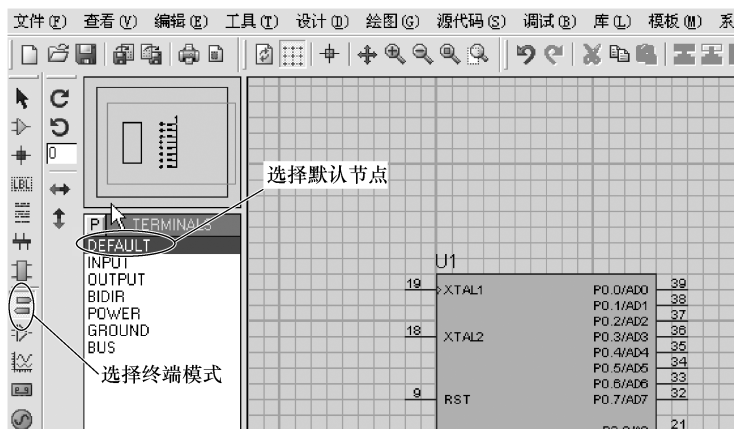


图 4-2　选择节点

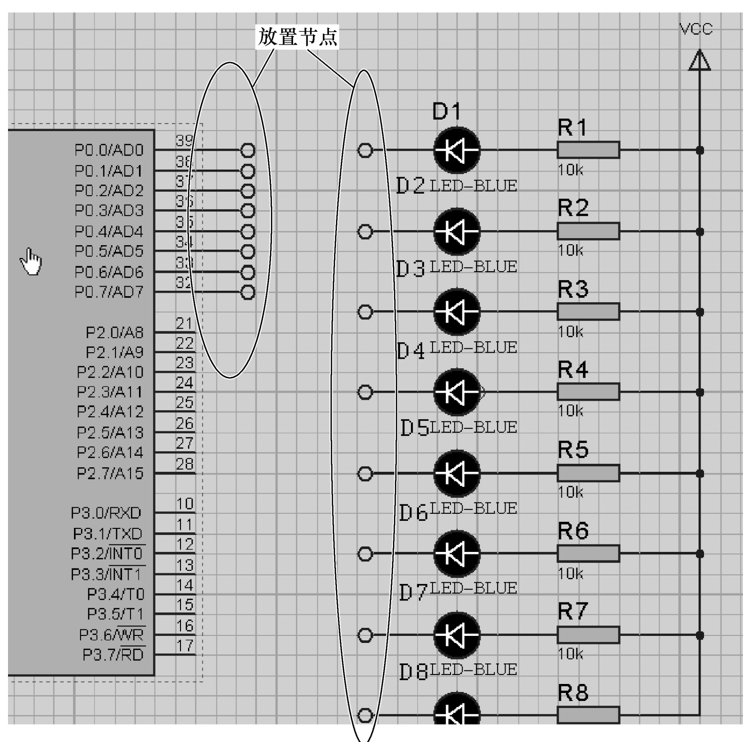


图 4-3　放置节点，通过节点连接

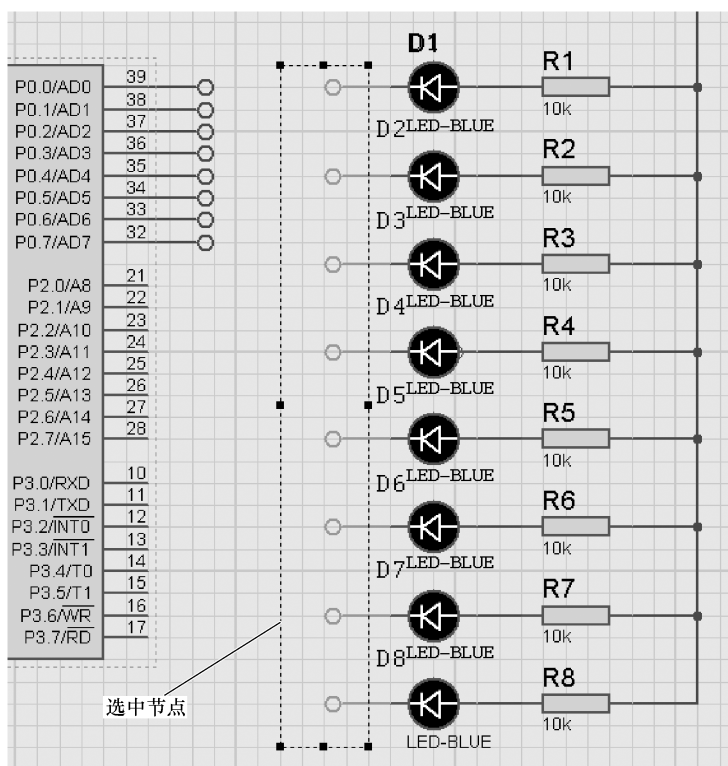


图 4-4　选中需要标注网络号的节点

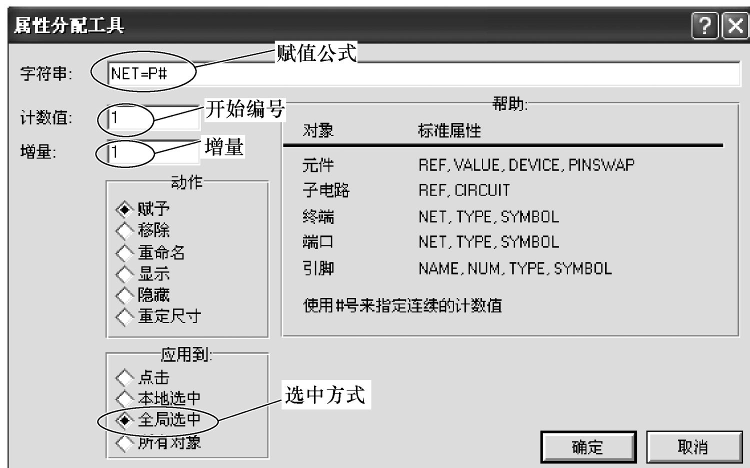


图 4-5 用属性设置工具标注网络号

同理，单片机端的 8 个节点也同样处理，使得每个节点与发光二极管的编号一一对应，如图 4-6 所示。

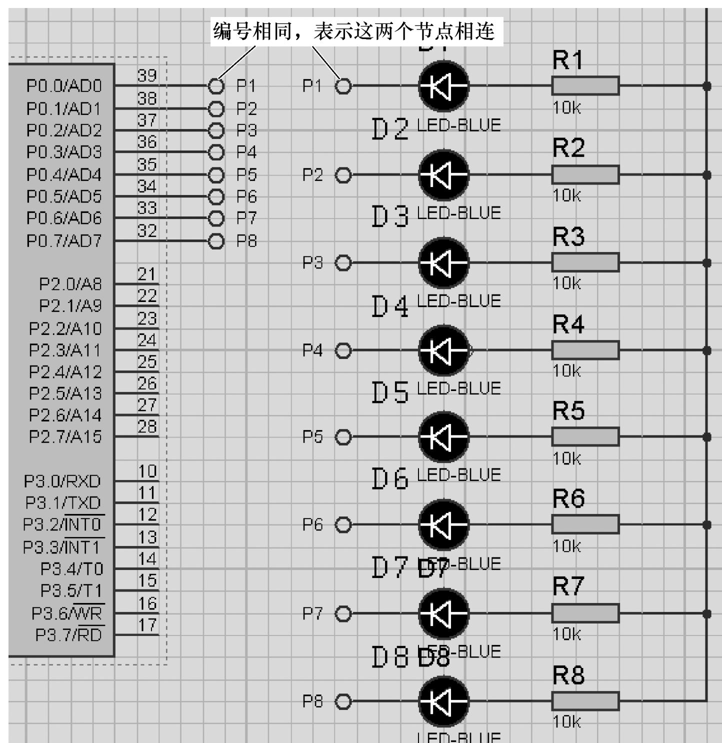


图 4-6 标注节点

8 个电阻也用属性设置工具把电阻的阻值改为 500Ω，如图 4-7 所示。

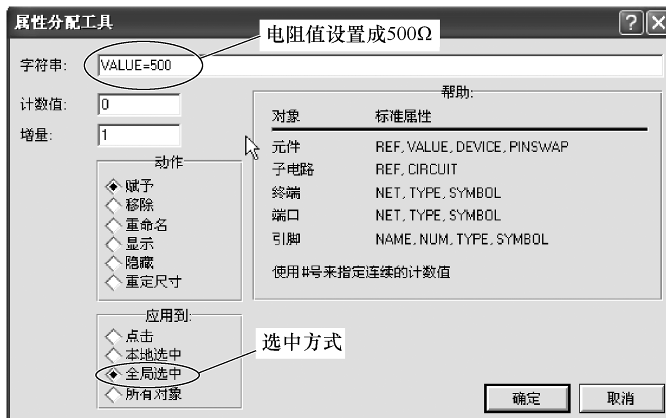


图 4-7　用属性设置工具修改阻值

这样，我们就把电路画完了，只要在单片机上加载可执行程序，就可以仿真了。

4.2　用总线连接发光二极管至单片机

前面我们用节点连接的方式，把发光二极管和单片机引脚连接起来。现在我们换一种方式，用总线的方式连接。修改电路，用光标选中所画节点，按“Delete”键，就删除了。

首先，我们来画一条总线。在模式工具栏选择“总线模式”，在元件列表窗口中选择“BUS”（即总线），就可以画了，如图 4-8 所示。

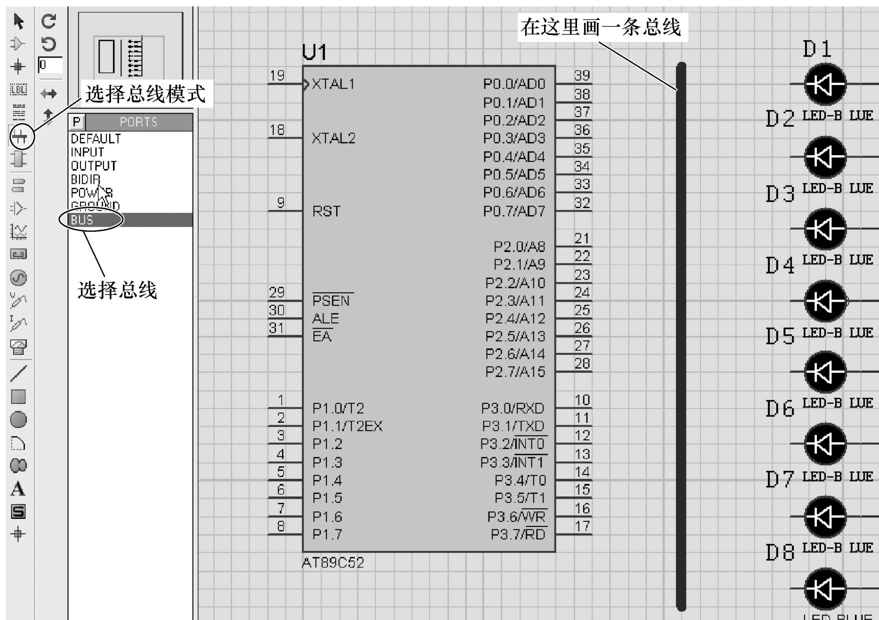


图 4-8　画总线

接着，用连线的方式连接到总线。把单片机引脚和发光二极管的连接点引向总线即可，如图 4-9 所示。注意：画斜线连接线需要把“自动连接器”按钮放开。

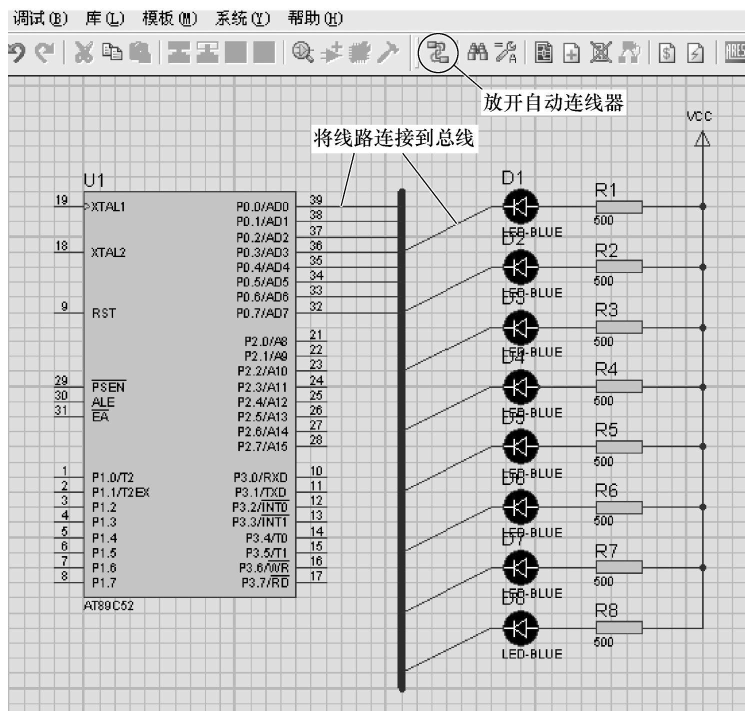


图 4-9 连接到总线

画完连接线以后，跟节点连接方式一样，还需要设置网络编号，才能将相关节点一一对应地连接起来。我们仍然采用“属性设置工具”来设置编号，这次采用逐个点击的方式编号。打开菜单“工具”→“属性设置工具”，设置参数与节点连接相仿，如图 4-10 所示。

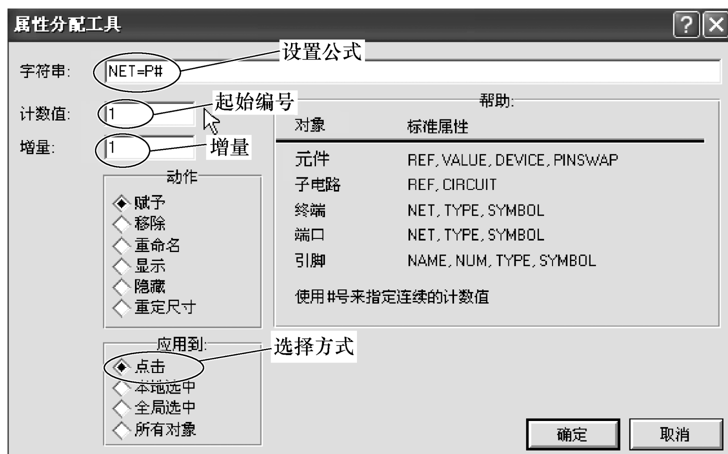


图 4-10 属性设置工具对话框

用鼠标左键单击“确定”按钮后，用鼠标左键逐个点击需要编号的连线。点一下，就

会在连线上自动按顺序编号，如图 4-11 所示。

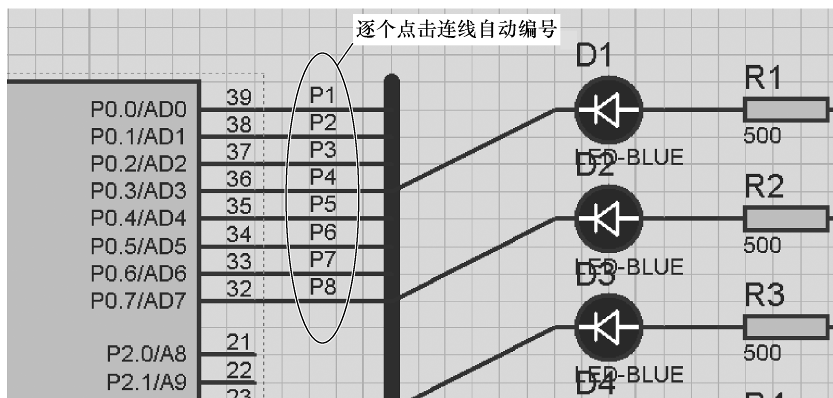


图 4-11　用属性修改工具设置总线网络编号

用同样的方法处理发光二极管至总线的编号，如图 4-12 所示。注意，属性设置工具的参数必须重新设置，才能从 P1～P8 重新开始。

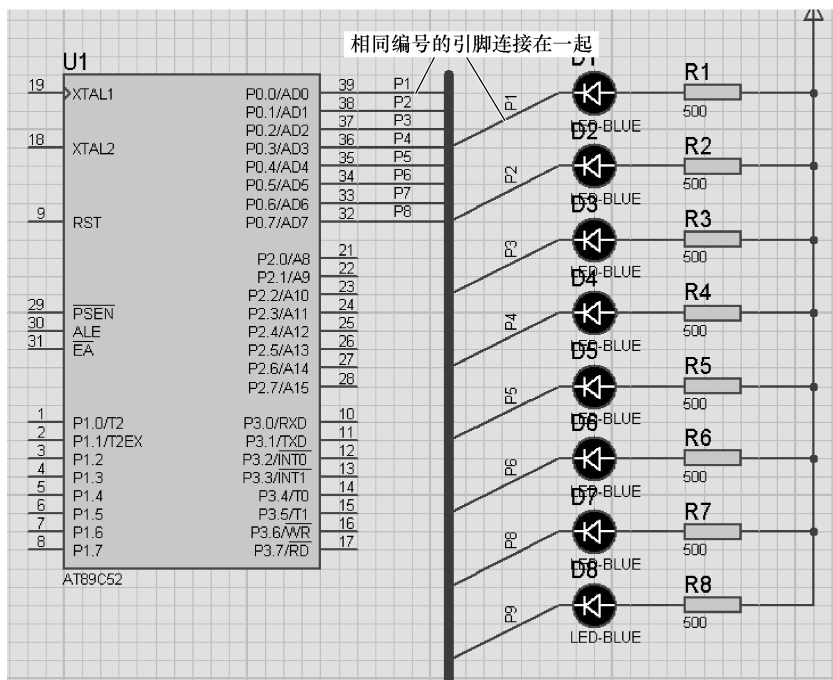


图 4-12　总线连接方法

这样，总线连接的工作就做完了，只要加载程序就可以仿真了。

4.3　程序设计和仿真运行

画好了原理图，接下去就是程序设计了。打开 Keil C51 uVision2 软件，新建一个工程

t4，如图 4-13 所示。

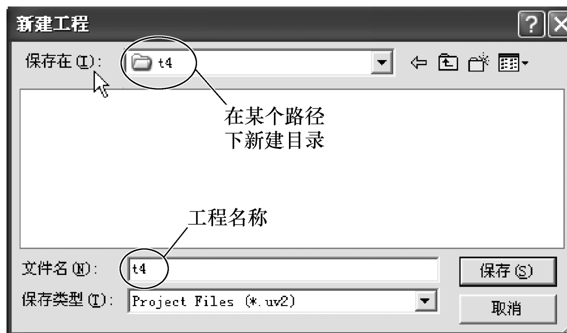


图 4-13　新建工程

接着是选择厂商和单片机型号 Atmel 和 89C52（参看第 3 讲），然后是新建文本文件，在其中编写或粘贴程序，程序内容如图 4-14 所示。

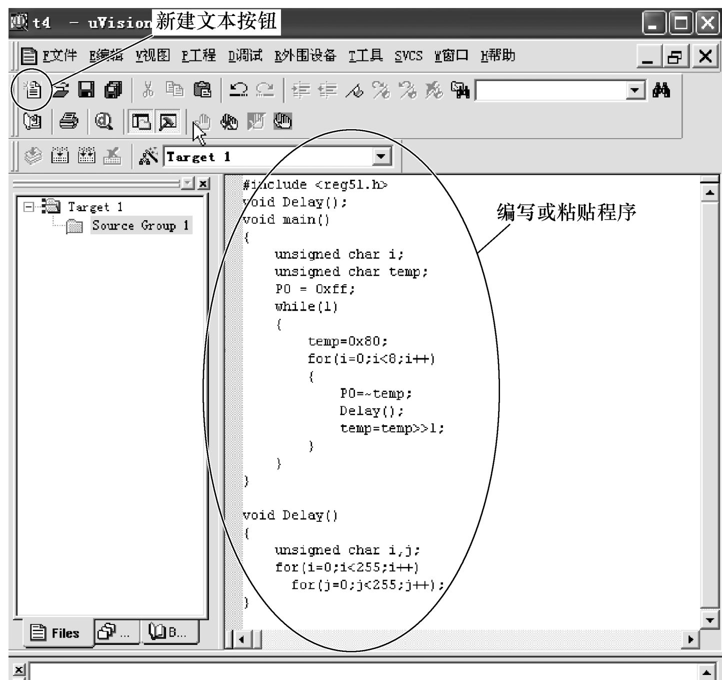


图 4-14　新建程序文件

将该文件命名为 t4. c 存盘，如图 4-15 所示。

将文件加入到工程中（参见第 3 讲），如图 4-16 所示。

接着，还要设置工程目标的属性，以便能编译生成可执行文件。将光标放在最上层“Target 1”上，选择菜单“工程”→“目标‘Target 1’属性”，如图 4-17 所示。

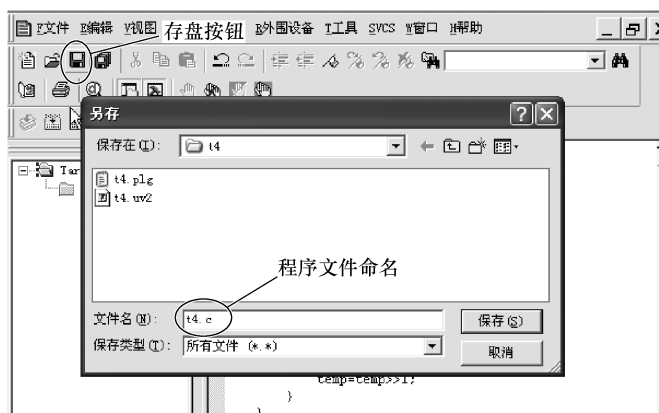


图 4-15　文件存盘

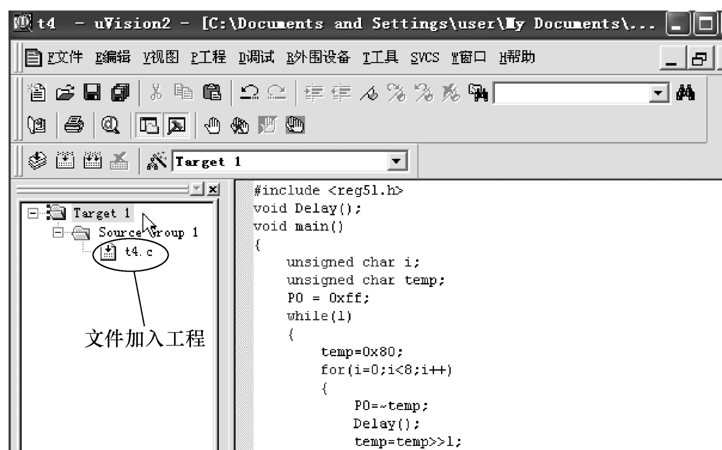


图 4-16　在工程中添加文件

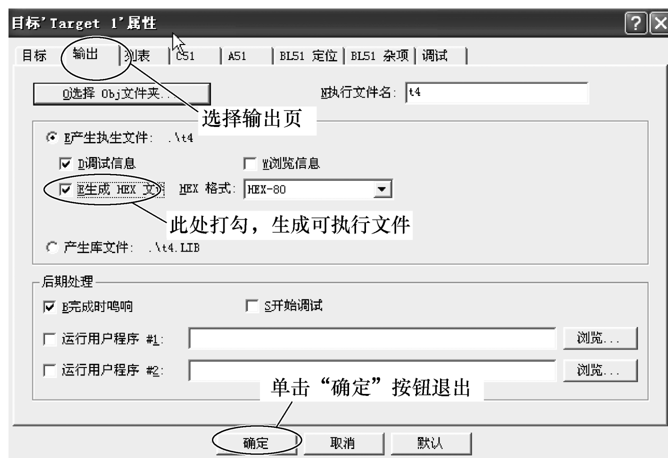


图 4-17　设置目标属性

这样，编译后，就会在工程目录中生成 t1. hex 可执行文件。

接下来，我们分析一下这个程序的结构和功能，先列出程序及说明：

```
#include <reg51.h>                  //单片机头文件
void Delay();                       //延时子程序说明
void main()                         //主程序
{
    unsigned char i;                //定义一个无符号字符 i
    unsigned char temp;             //定义一个无符号字符 temp
    P0 = 0xff;                      //端口 P0 赋值为 0xff
    while(1)                        //无限循环
    {
        temp =0x80;                 //变量 temp 赋值为 0x80
        for(i=0;i<8;i++)            //循环 8 次
        {
            P0 = ~temp;             //变量 temp 的值求反后赋值给端口 P0
            Delay();                //延时一段时间
            temp = temp >>1;        //temp 的值右移一位
        }
    }
}

void Delay()                        //延时子程序
{
    unsigned char i,j;              //定义无符号字符变量 i 和 j
    for(i=0;i<255;i++)              //外循环 255 次
      for(j=0;j<255;j++);           //内循环 255 次
}
```

程序由 3 部分组成：头文件、主程序和延时子程序。头文件是固定的，每个单片机程序都必须有。延时子程序使发光二极管在变化时有一个停留时间，通过一个外循环和内循环的空操作，共循环 255×255 次。这里需要说明一点，程序中把延时子程序放在最后，所以需要在头上说明一下，若子程序放在主程序之前，就不需要说明了。

下面我们来分析主程序。

我们要用单片机的 P0 口控制 8 个发光二极管循环点亮和关闭，就是要在 P0 口循环输出一个 0，也就是低电平。因为，在前面我们已经做了实验，发光二极管负极接单片机引脚，引脚输出低电平 0，发光二极管获得足够的电压降，就会发光，反之，单片机引脚输出高电平 1，则发光二极管不亮。P0 口 8 个引脚有几个低电平 0，就有几个发光二极管点亮。现在，我们需要只有一个 0，其他引脚都是高电平 1，并且需要循环变化，让这个 0 逐个从低到高或从高到低的走。可以用给单片机端口赋值的方法设置引脚的输出，如在开始，P0 =

0xff，就是给端口 P0 的所有引脚设置成高电平。0xff 就是十六进制 ff，折算成二进制就是 11111111，正好 8 个 1，每个十六进制对应二进制 4 位。因此，一开始是没有发光二极管点亮的。

首先是定义变量，i 用作循环。temp 作为输出的变量。我们来看无线循环中的循环体，先给 temp 赋一个初值 0x80，就是二进制 1000 0000，最高位是 1。在以下的 8 次循环中把 temp 求反后赋值给 P0 口，“~”就是求反的运算符。然后是延时一段时间，最后把 temp 值右移一位，转到下一个循环。见表 4-2。

表 4-2　temp 值与 P0 口输出值对照

循环次数	Temp 值（二进制）	P0 口值（二进制）
1	1000 0000	0111 1111
2	0100 0000	1011 1111
3	0010 0000	1101 1111
4	0001 0000	1110 1111
5	0000 1000	1111 0111
6	0000 0100	1111 1011
7	0000 0010	1111 1101
8	0000 0001	1111 1110

可以看到，P0 口的值有一个 0 从高到低的走，就使得引脚逐个变成低电平，也就控制了发光二极管逐个地点亮和关闭。用 temp 作为中间变量，是因为要使一个 0 从高到低地移动是困难的，但要使一个 1 移动却很容易，只要用“>>”就做到了，然后求反就可以了。

循环了 8 次后，temp 重新赋值为 1000 0000，继续下一个 8 次循环，继续不断地进行，就实现了发光二极管不断地从高到低的点亮和关闭。

接下来，我们编译程序，如图 4-18 所示。

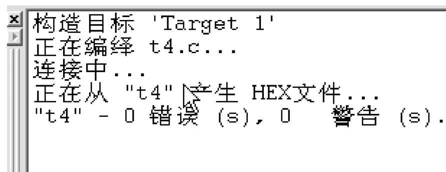


图 4-18　编译结果显示

显示 0 错误和 0 警告，说明编译成功。这样，我们就可以用编译出来的可执行程序进行仿真了。在电路中双击单片机，在弹出的“编辑元件”对话框的“Program File”栏，选择浏览按钮，找到可执行文件 t4. hex，如图 4-19 所示。

单击“确定”按钮来保存，单击仿真按钮进行仿真运行，如图 4-20 所示。

我们实现了发光二极管从高到低（P0^7 ~ P0^0）的变化点亮和关闭，那么，如何使它从低到高的变化呢？刚才我们给 temp 赋值是 0x80，最高为为 1，只要把它改成 0x01，即最低



图 4-19 浏览查找可执行程序

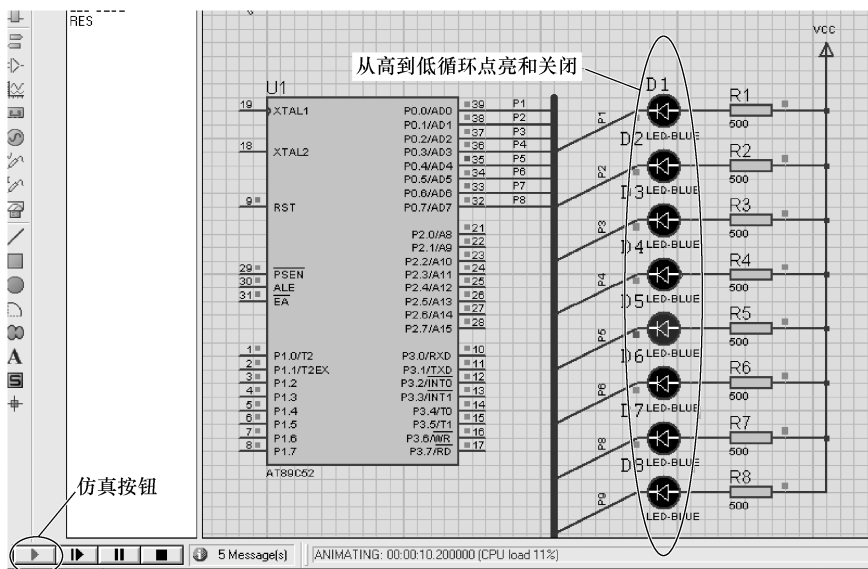


图 4-20 流水灯控制

为 1，求反后就是最低位为 0。另外，还要把移动方向改为向左移动“ < < ”。这样，变化就是从低到高的点亮和关闭。请读者自己试一下，看能否改过来！只要把程序改动一下，然后重新编译。电路中单片机程序加载不需要重做，很方便吧！

第5讲 1位数码管计数器

 导读

这一讲我们要介绍数码管、按键的应用，并涉及单片机中断处理技术。在计算机应用中有输入输出技术，发光二极管就是一种输出，表示一种运算结果或状态。数码管也是一种输出设备，能输出数字和部分字符，在单片机应用中非常广泛。按键则是一种输入，也是一种人与计算机交互的手段，按键是最基本的输入手段。中断是计算机处理外部或内部突发时的一种方法，往往用于处理人机交互事件，使程序更加清晰、层次分明。

5.1 1位数码管计数电路

我们先用一个数码管，搭建一个简单的计数电路。数码管其实就是由7个发个二极管组成的。其中某些管子亮，就构成了一个数字或字母。图5-1所示为数码管的结构图。

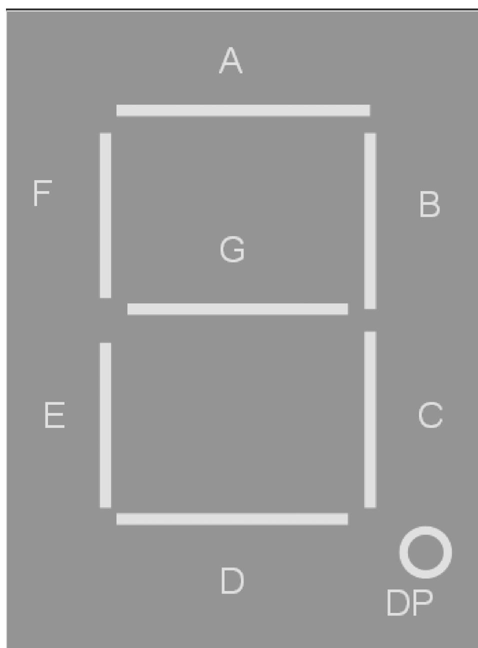


图5-1 数码管结构图

从图 5-1 中可以看出，如果 B、C 亮，就是数字 1，如果 A、B、C 亮，就是数字 7。所以，需要有一个表示数字或字符的码段表，根据需要输出的数字或字符确定哪几位管子要点亮。有时还需要有一点，就是右下角的 DP，这样共需要 8 个管子。发光二极管需要接电源和接地，这就有一个如何将 8 个管子接入电源和地的问题。有两种数码管，根据接法不同分为共阴接法和共阳接法。即 8 个管子公共端接地，称为共阴接法的数码管，而 8 个管子公共端接电源，称为共阳接法的数码管。两种数码管的原理图如图 5-2 所示。

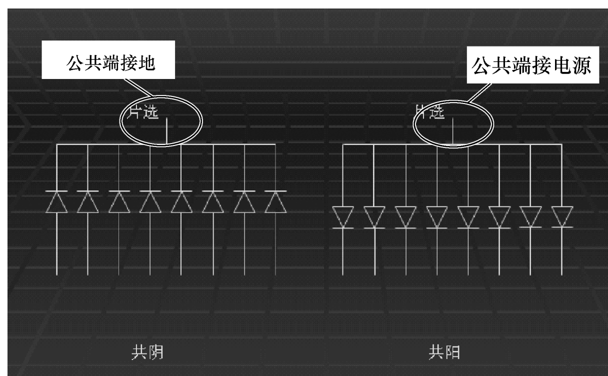


图 5-2 两种数码管原理图

这个公共端也是控制数码管是否启用的引脚，称为“片选”，即这片电路是否被选中。由于接法的不同，共阴和共阳数码管的码段表也是不同的，表 5-1 是两种数码管的码段表：

表 5-1 数码管码段表

字符	共阴顺序，小数点暗		共阳顺序，小数点暗	
	dp g f e d c b a	十六进制数	dp g f e d c b a	十六进制数
0	0 0 1 1 1 1 1 1	3F	1 1 0 0 0 0 0 0	C0
1	0 0 0 0 0 1 1 0	06	1 1 1 1 1 0 0 1	F9
2	0 1 0 1 1 0 1 1	5B	1 0 1 0 0 1 0 0	A4
3	0 1 0 0 1 1 1 1	4F	1 0 1 1 0 0 0 0	B0
4	0 1 1 0 0 1 1 0	66	1 0 0 1 1 0 0 1	99
5	0 1 1 0 1 1 0 1	6D	1 0 0 1 0 0 1 0	92
6	0 1 1 1 1 1 0 1	7D	1 0 0 0 0 0 1 0	82
7	0 0 0 0 0 1 1 1	07	1 1 1 1 1 0 0 0	F8
8	0 1 1 1 1 1 1 1	7F	1 0 0 0 0 0 0 0	80
9	0 1 1 0 1 1 1 1	6F	1 0 0 1 0 0 0 0	90
A	0 1 1 1 0 1 1 1	77	1 0 0 0 1 0 0 0	88
B	0 1 1 1 1 1 0 0	7C	1 0 0 0 0 0 1 1	83
C	0 0 1 1 1 0 0 1	39	1 1 0 0 0 1 1 0	C6
D	0 1 0 1 1 1 1 0	5E	1 0 1 0 0 0 0 1	A1
E	0 1 1 1 1 0 0 1	79	1 0 0 0 0 1 1 0	86
F	0 1 1 1 0 0 0 1	71	1 0 0 0 1 1 1 0	8E

因为数码管中发光二极管与一般的发光二极管一样，需要控制电流，也需要接电阻，阻值大小约 200～500Ω。下面我们就用 Proteus 的 ISIS 来构建电路。首先，选择元件单片机、电阻和数码管。我们选用共阴数码管，元件名称是 7SEG-COM-AN-GRN，意思是 7 段、共阳、绿色数码管。元器件列表见表 5-2。

表 5-2　元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	7SEG-COM-AN-GRN	数码管	共阳、绿色
2	BUTTON	按键	
3	RES	电阻	标准，阻值可设定
4	89C52	单片机	51 系列

将数码管连接在单片机 P0 口，并接上电阻，阻值均为 500Ω，数码管共阳端接电源（始终选中）。为了使电路简洁，去掉元件的说明（菜单“模板”→“设计默认值”，去掉“显示隐藏文本”的勾选，如图 5-3 所示）。

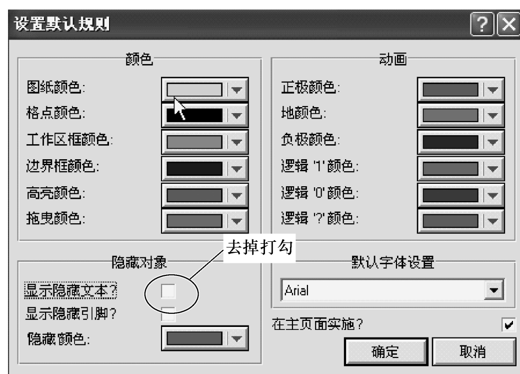


图 5-3　去掉元件说明设置

将连接到数码管的电阻值用属性设置工具修改为 500Ω，如图 5-4 所示。



图 5-4　用属性设置工具修改电阻值

因为这个数码管没有小数点，只要 7 个电阻就够了。接上数码管，在共阳端接电源，如

图 5-5 所示。

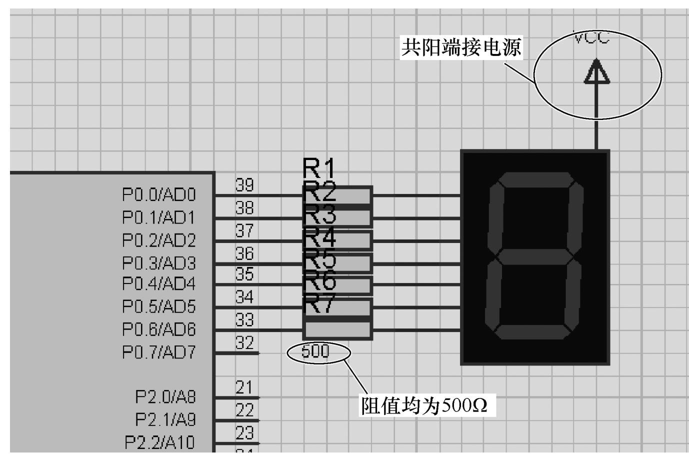


图 5-5　一位数码管电路

这样，电路图就画完了。需要说明一点，实际电路图中需要加上晶振和复位电路，在这里都省略了，在正式制作电路板前需要补上。

5.2　按钮与中断处理

接着，我们来设计按钮处理。在软件处理上，按钮与刀开关差不多。按钮按下后会自动弹起，开关则不会。按钮的元件名称是 BUTTON，它的一端接单片机的引脚，另一端接地。单片机的引脚会有 3 种情况，接高电平、开路（高阻）和接低电平。一般把接高电平和开路（高阻）一样看待，所以只有两种情况了，即高电平和低电平。当按钮接单片机引脚时，开路就是高电平，短路（按下）就是接地了，即低电平。因此，单片机判断按钮是否按下，就看接按钮的引脚是否是低电平。在元件窗口添加按钮 BUTTON，如图 5-6 所示。

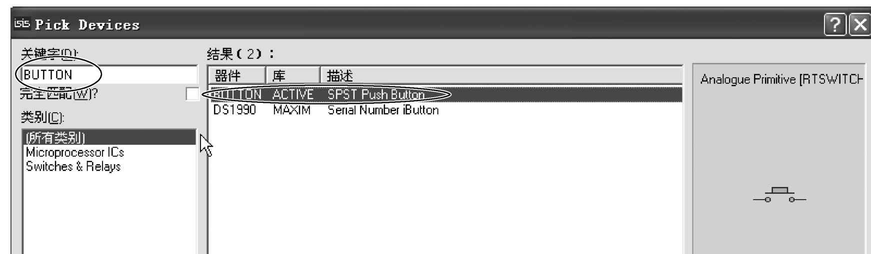


图 5-6　按钮选择

在单片机 P3.2 和 P3.3 引脚分别接一个按钮，按钮一端接地，如图 5-7 所示。

为什么要接在 P3.2、P3.3 引脚上呢？这是因为，这两个引脚是单片机的外部中断引脚，只有这两个引脚才能引起外部中断，在下面程序中会用到。关于中断的处理比较复杂，我们可以先套用基本的格式，而不追究它的原理。

什么是中断处理呢？打个比方，我们常规的工作是看书。在看书的时候来了电话，打断了看书，这就是中断。那么，接电话就是中断处理。接完电话继续看书，就相当于程序中的

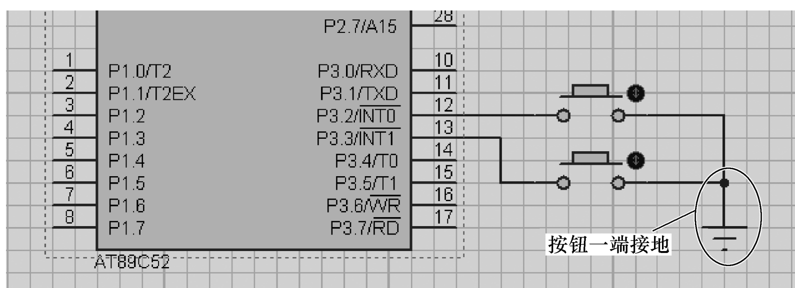


图 5-7　按钮处理电路

返回主程序。这里的主程序就是看书，中断处理程序就是接电话。

89C52 单片机共有 5 个中断源，2 个外部中断，2 个定时/计数中断，和一个串口中断（用于通信）。外部中断的引脚是 P3.2、P3.3，定时/计数的引脚是 P3.4、P3.5，串口通信中断引脚是 P3.6、P3.7。

为了实现中断控制，需要设置一些中断控制寄存器。通过这些寄存器来控制单片机的行为，这些设置都是通过程序实现的。外部中断至少需要用到两种寄存器：定时/外部中断控制寄存器 TCON，和中断允许控制寄存器 IE。

先看定时/外部中断控制寄存器 TCON：

1）高 4 位用于定时器控制（D4 ~ D7）；

2）底 4 位用于外部中断控制（D0 ~ D3）；

3）IT0 ：外部中断 0 类控制，通过软件设置或清除；

4）IE0：外部边沿触发中断 0 类控制，由片内硬件自动清零；

5）IT1：外部中断 1 类控制，性质同 IT0；

6）IE1：外部边沿触发中断 1 类控制，性质同 IE0。

再看中断允许控制器 IE：

1）EX0：外部中断 0 控制位（EX0 = 1 允许中断，EX0 = 0 禁止中断）；

2）ET0：定时/计数 T0 中断控制位（ET0 = 1 允许，ET0 = 0 禁止）；

3）EX1，ET1：外部中断 1 和定时/计数 T1 中断控制位，性质同 EX0、ET0。

4）ES：串口中断控制位，ES = 1 允许，ES = 0 禁止；

5）EA：中断总控制位，EA = 1 开发，EA = 0 禁止；

最后就是中断处理程序的写法了，我们看一下定义，见表 5-3。

表 5-3　中断处理编号

中断编号	中　断　源
0	外部中断 0
1	定时/计数 0 溢出中断
2	外部中断 1
3	定时/计数 1 溢出中断
4	串行口中断

■ 定义中断服务函数

■ 函数类型　函数名（形式参数）[interrupt n] [using m]

■ n： 中断编号 m： 寄存器组号

■ 例：void intersvr0（void）interrupt 0 using 1

看起来确实有点复杂，一下子看不懂。没有关系，等看了实例以后，慢慢就理解了。

5.3 程序设计与仿真

下面我们就开始设计数码管计数的程序。首先，设计一个单独的计数程序，即不涉及按键，只是数码管自动地计数，从 0 到 9 循环计数，每隔一个时间间隔加一个数，到达 9 后再从 0 开始。这个时间间隔也是普通的延时，没有采用精确定时的方式。采用精确定时的方法可以用单片机提供的定时计数方法做，我们在以后讲解。

这个程序的思路很简单，设置一个变量，让它从 0 到 9 变化，并逐个送到数码管显示。在送到数码管前，先要把数字变成相应的数码管码段值，才能在数码管显示正确的数字。所以，把数字变成相应的码段值是程序的关键。以下我们列出程序及说明：

```
#include <reg51.h>                    //单片机头文件
unsigned char code Tab[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                      //共阳数码管码段表
void Delay()                          //简单延时子程序
{
  unsigned char i,j;
  for(i=0;i<255;i++)
    for(j=0;j<255;j++);
}

void main()                           //主程序
{
  unsigned char i=1;                  //初值为 1
  while(1)
  {
    P0=Tab[i];                        //送相应码段表值
    Delay();                          //调用延时子程序
    i++;                              //变量加 1
    if(i>9) i=0;                      //若变量大于 9,重新设置为 0
  }
}
```

除了头文件外，首先定义一个数组，将数码管的码段表存放数组中，以便调用。按照数组的下标，第 0 个值就是数字 0 的码段，第 1 个值就是数字 1 的码段，依次类推。因此，P0 = Tab [i]，就把 i 这个数的码段送到端口 P0，使该端口按照码段的值输出电平，从而控制数码管显示 i 变量中的数字。也就是说，P0 = Tab [0]，就会在数码管显示 0，P0 = Tab [9]，就会在数码管显示 9。

在无限循环中，除了向 P0 端口发送码段表值、延时以外，还要控制变量 i，让它在 0 到 9 之间变化。

接着，我们用 Keil C51 新建工程、新建程序文本、添加程序至工程、设置工程的输出选项可以生成可执行文件、编译工程，直到生成可执行文件，并加载到单片机中（若还不熟悉，请参考前面几讲）。图 5-8 所示为加载程序后的仿真运行情况。

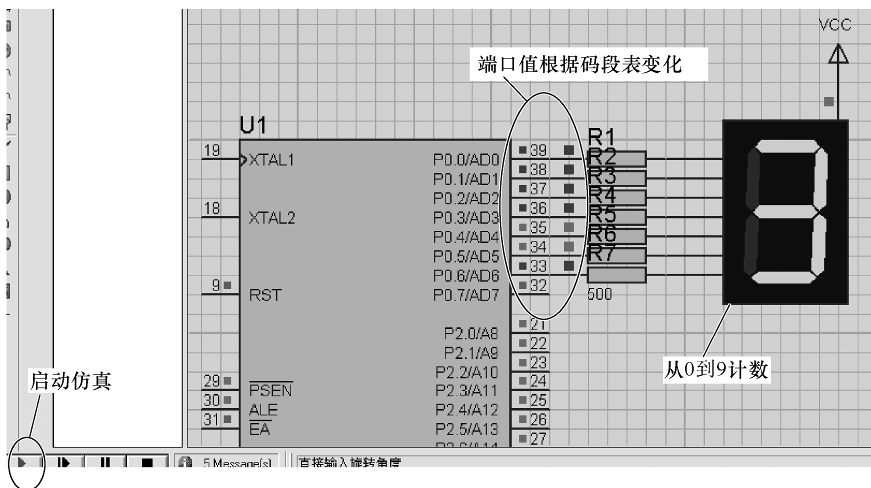


图 5-8 一位数码管自动计数电路

然后，我们来设计采用按钮计数的程序。首先，我们不用中断处理的方式来实现按钮加数和减数的程序。程序及说明如下：

```
#include <reg51.h>                    //单片机头文件
unsigned char code Tab[ ] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                      //共阳数码管码段表
unsigned char p = 0;                  //计数变量
sbit S1 = P3^2;                       //定义按钮 S1
sbit S2 = P3^3;                       //定义按钮 S2

void delays(void)                     //延时子程序
{
  int i,j;
  for(i = 300;i > 0;i--)
        for(j = 100;j > 0;j--);
}

void main(void)                       //主程序
{
  while(1)                            //无限循环
  {
```

```
        P0 = Tab[p];                    //输出码段表
        if(S1 ==0)                      //如果 S1 按下
        {   p + +;                      //计数加 1
          delays();                     //调用延时子程序
          if(p >9)                      //如果计数大于 9
            p =9;                       //计数 =9
        }
        if(S2 ==0)
        {   p--;                        //计数减 1
          delays();                     //调用延时子程序
          if(p <1)
            p =1;                       //计数 =1
        }
    }
}
```

这个程序与上一个程序不同的地方是，把自动计数改为手动计数。根据用户按钮的不同增加或减少计数。首先定义 2 个按钮 S1 和 S2，分别与 P3.2 和 P3.3 相连。判断按钮是否按下就是看该引脚是否为低电平，若 S1 ==0，即按钮 1 按下，就将计数值 p 加 1，并判断 p 是否超过 9，因为只有一个数码管，最大计数值为 9。因此，若超过 9 就把计数值 p 改为 9。

同理，判断 S2 是否按下，若 S2 ==0，即按钮 2 按下，就将计数值 p 减 1，并判断 p 是否小于 1，若小于 1 就把计数值 p 改为 1。这样，就限定了计数值在 1 ~9 之间变化。

循环体的工作就是，输出选定（p 值）的码段表值，判断按钮是否按下，根据不同的按钮，增加或减少计数值，在下一次循环输出。图 5-9 所示为加载程序后的仿真运行情况。

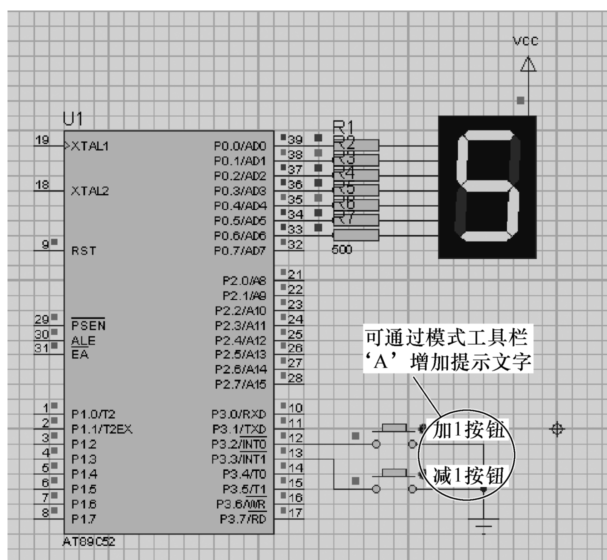


图 5-9　一位数码管手动计数的电路

从这个程序看出，主程序的主要工作就是判断按钮是否按下，并作出增加计数或减少计数值。若还有其他工作要做，就会增加程序的复杂度和开发难度。是否有更好的方法，使程序开发更简洁、清晰，这就是中断处理的方法，下面我们就来看一下用中断处理的方式实现上述功能。

```
#include <reg51.h>                    //单片机头文件
#define uchar unsigned char           //简化变量定义,将 unsigned char 简化为 uchar
unsigned char code Tab[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                      //共阳数码管码段表
uchar p = 1;                          //定义计数变量

void delays(void)                     //延时子程序
{
    uchar i;
    for(i = 300;i > 0;i--);
}

void main(void)                       //主程序
{
    EA = 1;                           //开总允许开关
    IT0 = 1;EX0 = 1;                  //开外部中断 0 和外部中断 0 允许分开关
    IT1 = 1;EX1 = 1;                  //开外部中断 1 和外部中断 1 允许分开关
    while(1)                          //无限循环
    {
    P0 = Tab[p];                      //输出码段表
    delays();                         //延时
    }
}

void intersvr0 (void) interrupt 0 using 1  //外部中断 0 处理子程序
{
    p++;
    if(p > 9)
     p = 9;
}

void intersvr1 (void) interrupt 2 using 1  //外部中断 1 处理子程序
{
    p--;
    if(p < 1)
```

```
    p = 1;
}
```

与上一个非中断处理程序相比，主程序的循环体变得非常简洁，只有输出码段表和延时。在程序的开始，也没有开关引脚的定义，这是因为外部中断引脚是内定的。但是在主程序开始增加了有关中断处理 TCON 和 IE 寄存器的设置，一是打开总允许开关，EX = 1。若 EX = 0 就是关闭总允许。二是打开 2 个外部中断和分允许开关，IT0 = 1 就是打开外部中断 0，EX0 = 1 就是打开外部中断 0 分允许开关。同理，IT1 = 1 就是打开外部中断 1，EX1 = 1 就是打开外部中断 1 分允许开关。这些设置是使用中断处理的前提，也可以在程序中进行变化。

再来看 2 个中断处理子程序，它的内容就是上一个程序中的判断语句中的内容，增加或减少计数值 p 的值。中断处理子程序的头部写法是有规定的，其中 interrupt 0 是外部中断 0 的子程序，interrupt 2 是外部中断 1 的子程序，不能更改，全部中断编号的定义见上一节的中断编号表，外部中断是 0 和 2，定时/计数中断是 1 和 3，串口中断是 4。

从结构上看，中断处理程序更加简洁，设置也不难，只要按规定设置参数和编写中断子程序的头部就可以了。

第6讲

4 位数码管计时器

 导读

这一讲我们要介绍多位数码管的电路设计和程序设计，并要讲解单片机中关于定时器的用法。多位数码管的接法与一位数码管相似，就是多了一个片选电路，以确定选择哪个数码管进行工作。单片机定时器的应用也与中断处理相似，需要设置一些规定的寄存器，以此来规定单片机的工作状态。

6.1 4 位数码管计时器电路

选择多位数码管可以选择多个一位数码管，也可以选择多位合一的数码管，这样电路就会比较简洁。现在，我们选择一个 4 位的数码管作为计时电路，元器件列表见表 6-1。

表 6-1 选择元器件列表

序号	元器件名称	名称说明	备注
1	7SEG-MPX4-CA	4 位数码管	共阳、红色
2	RES	电阻	标准，阻值可设定
3	89C52	单片机	51 系列

用以上元器件画出原理图，如图 6-1 所示。

同一位数码管一样，为了限流，需要在数码管接入单片机之间接限流电阻，阻值为 500Ω。从电路图我们看到，4 位数码管共享一组数据线，通过片选线来确定由哪一位输出显示。若要使用小数点，可以把 DP 数据线接上。修改 7 个电阻阻值仍然可以用“属性设置工具”完成。若用多个一位数码管，接法也是这样，把所有的码段数据线接同一个单片机端口即可，通过片选线区分每个数码管（通过程序实现）。这个 4 位数码管的共阳端是默认接好的，所以不用另作处理。

从这个多位数码管电路我们得出一个结论：码段表数据线共享，片选线分别连接，最好连接到一个端口，以便于编程处理。

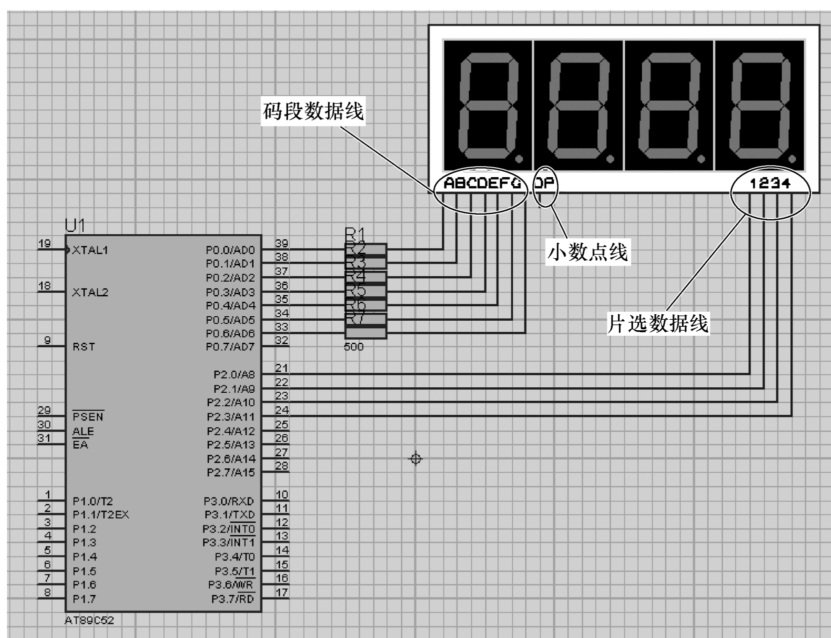


图 6-1　4 位数码管电路

6.2　多位数码管显示程序设计

首先，我们要解决如何在多位数码管显示数据的问题。回忆一位数码管的显示输出：定义码段表数组，通过数组下标值，调用相应码段至单片机端口输出。多位数码管显示输出与一位数码管相似，也是先要定义数码管的码段表，并通过数组下标调用输出。但是，由于多个数码管的码段数据线都是连在一起的，只有用片选线才能区分每位数据输出。

另外，有多位数据，例如，有 4 位数据，数据的存放问题也需要专门考虑。这里，我们仍然采用数组的方式存放 4 位数据，因为数组在程序处理上很方便。这样，一个数组位，就相当于一个变量，存放着一位数据，再把这个数组位作为码段表数组的下标，调用出相应的输出码段值送至单片机端口。

下面，我们编写一个 0 ~ 500 的自动计数程序，共用到 3 位数码管。程序从 0 开始计数显示，直到 500 后，再回到 0，循环往复进行。

```
#include <reg51.h>                                          //单片机头文件
unsigned char code Tab[ ] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
                             0x80, 0x90};                   //共阳数码管码段表
unsigned char Dat [ ] = {0, 0, 0, 0};                       //存放 4 位数据数组

unsigned char tmp, i, j;                                    //定义临时变量、循环变量
unsigned char S = 0;                                        //定义计数变量
void Delay (int n)                                          //延时子程序
```

```
{
  int i;
  for (i = 0; i < n; i ++);
}

void main ()                                 //主程序
{
  while (1)                                  //无限循环
  {
    S ++;                                    //计数值加 1
    if (S > 500)                             //如果计数值大于 500 则清零
        S = 0;
    Dat[0] = S/100;                          //取计数值的百位数
    Dat[1] = S%100/10;                       //取计数值的十位数
    Dat[2] = S%10;                           //取计数值的个位数

    tmp = 0x01;                              //片选初值
    for (i = 0; i < 3; i ++)                 //循环 3 次
    {
     P2 = tmp;                               //将片选值送 P2 口
       P0 = Tab[Dat[i]];                     //将某一位码段值送 P0 口
     tmp = tmp << 1;                         //片选值移至下一位
     Delay (250);                            //调用延时子程序
    }

    for (i = 0; i < 150; i ++)               //延时一段时间
      for (j = 0; j < 100; j ++)
          Delay (60000);                     //调用延时子程序

  }
}
```

这个程序的头文件和数码管的码段表定义都和一位数码管显示一样，但增加了多位数存放和数字分离的内容。这里用一个数组 Dat [ ] = {0，0，0，0} 存放 4 位数，初值都为 0。用变量 S 作为计数，tmp 作为片选值。

数字分离的方法很奇妙，百位数分离用 Dat [0] = S/100 公式解决，因为这里 S 值不会超过 500，所以不用担心结果会大于一位数。例如，S = 235，则 S/100 = 2，Dat [ ] 是整数，小数部分自动删除。十位数分离用 Dat [1] = S% 100/10 公式解决，S% 100 取除 100 的余数，再把这个余数除 10，就得到十位数了。例如，还是以上数字 S = 235，S% 100/10 = 3。最后是个位数的分离，用 Dat [2] = S% 10，取 S 除以 10 的余数，很容易理解。

这里 Dat［0］存放百位数，Dat［1］存放十位数，Dat［2］存放个位数。在输出时要按从高到低的顺序输出，不能搞错，否则数字就反了。在输出时，先选定片选值，即决定在哪个数码管输出，送到 P2 口，因为我们将片选线接在 P2 口。第 1 个片选值是 1，根据数码管的标识在最左边位输出，然后通过 P0 口输出码段值，这时，就会在第 1 位显示数据了。接着，把片选值向左移动一位，为下一位输出作准备，停留片刻后进行下一位输出。这样，共循环 3 次，将 3 位数都输出显示在数码管上了。因为，我们的眼睛有视觉残留，所以感觉 3 位数字一起显示出来。但由于在输出完毕后还要停留较长时间才会进入下一个数的显示，输出会有闪烁的感觉。

这里，延时子程序采用参数的方式，可以调节延时的长短。数码管输出移位延时比较短，其他延时可以比较长，这样就能用一个延时子程序解决多种需要了。仿真结果如图 6-2 所示。

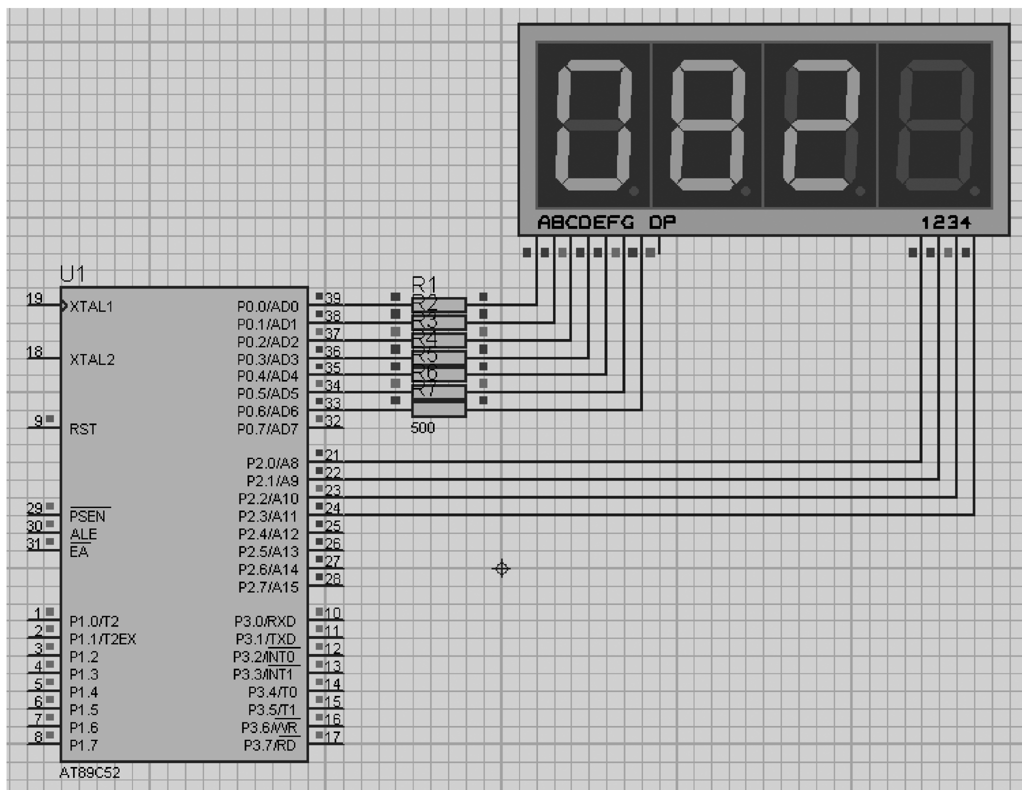


图 6-2　多位数码显示仿真结果

6.3　定时器原理和程序设计

我们已经实现了用多位数码管计数显示的问题。以下，我们来研究定时计数的问题，即显示时间的问题。在单片机中具有定时器的功能，当然，需要编程才能实现定时，并且有一点复杂。与前面介绍的外部中断功能一样，也是通过设置寄存器来规定单片机状态的。在介

绍外部中断时，我们已经知道，定时/计数的中断编号是 1 和 3，即定时/计数中断处理程序的头部必须是 interrupt 1 或 interrupt 3。除此之外，还需要知道什么呢？我们先看一下单片机定时器的结构如图 6-3 所示。

1）内设 2 个 16 位可编程定时/计数器 T0，T1；

2）具有计数方式和定时方式；

3）4 种工作模式。

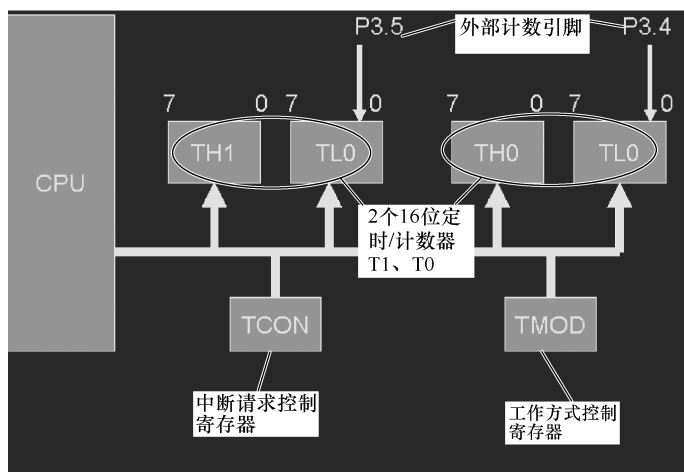


图 6-3　单片机定时器结构图

再看一下中断请求控制寄存器 TCON 的定义见表 6-2。

表 6-2　中断请求控制寄存器 TCON 的定义

TCON 位	D7	D6	D5	D4	D3	D2	D1	D0
位名称	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
功能	T1 中断标志	T1 启动控制	T0 中断标志	T0 启动控制	INT1 中断标志	INT1 触发方式	INT0 中断标志	INT0 触发方式

1）高 4 位定时计数，低 4 位外部中断。

2）TF1：当 T1 被允许计数后，从初值开始加 1 计数，至最高位溢出，TF1 置 1，表示中断请求。CPU 响应中断请求后对 TF1 清零。

3）TR1：定时计数器控制位，TR1 =1 启动 T1 运行，TR1 =0 时则 T1 停止运行。

4）TF0：T0 溢出标志，意义同 TF1。

5）TR0：T0 运行控制位，意义同 TR1。

工作方式控制寄存器 TMOD 的定义见表 6-3。

表 6-3　工作方式控制寄存器 TMOD 的定义

高 4 位控制 T1				低 4 位控制 T0			
门控位	定时/计数方式	工作方式		门控位	定时/计数方式	工作方式	
GATE	C/T	M1	M0	GATE	C/T	M1	M0

1）TMOD 高 4 位对 T1，低 4 位对 T0；

2）M1，M0：工作方式选择；

3）C/T：计数/定时方式选择位；

■ C/T = 1　对外部事件脉冲计数，负脉冲有效；

■ C/T = 0　对片内机器周期脉冲计数，作定时器；

4）GATE：门控位；

■ GATE = 0　只受 TCON 中 TR 的控制；

■ GATE = 1　同时受 TR 和外中断输入信号双重控制；

例 1：设置定时器 T1，工作方式 1

TMOD = 0x10；

例 2：设置定时器 T0，工作方式 2

TMOD = 0x02；

工作方式选择列表见表 6-4。

表 6-4　工作方式选择

M1	M0	方　式	功 能 说 明
0	0	0	13 位定时器（TH 的 8 位和 TL 的低 5 位）
0	1	1	16 位定时器/计数器
1	0	2	自动重装初值的 8 位计数器
1	1	3	T0 分成两个独立的 8 位计数器，T1 在方式 3 停用

1）工作方式 0：13 位计数结构，TH 全部 8 位，TL 低 5 位，TL 的高 3 位未使用。中断后计数清零，需重新设置。

2）工作方式 1：16 位计数结构，TH 和 TL 均为 8 位。中断同上。

3）工作方式 2：具有自动装载功能，TL0 为计数器，TH0 为预置寄存器，计数结构只有 8 位。

4）工作方式 3：2 个独立的 8 位计数器，TL0 既可作计数器，也可作定时器。TH0 只能作定时器。

要使用定时器，首先要计算定时器的初值，计算公式和一般方法如下：

1）T（初值）= 2^N − 定时时间/机器周期时间

2）N 与工作方式有关

■ 方式 0　　N = 13

■ 方式 1　　N = 16

■ 方式 2、3　　N = 8

3）1 机器周期时间 = 12/fosc = 12/12MHz = 1μs

例：晶振 12MHz，求方式 1，T0 定时 0.2ms 初值

2^{16} − 200/1 = 65536 − 200 = FF38H

则，高位 TH0 = FFH，低位 TL0 = 38H

看了以上材料，你是否觉得定时器太复杂了。不要着急，看一下实例，然后对照前面介

绍的表格，很快就能熟悉它的用法。

例：用 T0，方式 1，计时 0.05s，设计一个发光二极管闪烁（0.05s 开关一次）程序。

计算初值

- 高位 TH0 =-50000/256；
- 低位 TL0 =-50000%256；

为什么不用 $2^{16}-50000$ 呢？因为-50000 的补码等于 $2^{16}-50000$

定时计数器 0 的中断号为 1，所以，定时器的中断处理子程序头部为

- Void intsrv1(void) interrupt 1 using 1
- 中断后需重新定义初值（方式 0，1）

主程序和中断处理子程序如下：

```
#include <reg51.h>                   //单片机头文件
sbit LED = P0^0;                      //定义发光二极管引脚
void main (void)                      //主程序
{
  EA = 1;                             //允许所有中断
  ET0 = 1;                            //允许 T0 中断
  TMOD = 0x01;                        //T0 方式 1 计时 0.05s
  TH0 = -50000/256;                   //定时器 T0 的高八位
  TL0 = -50000%256;                   //定时器 T0 的低八位
  TR0 = 1;                            //启动 T0 定时器
  for (;;);                           //无限循环
}
/* 定时器 0 中断服务子程序，中断号 1   */
void intserv1 (void) interrupt 1 using 1
{
  TH0 = -50000/256;                   //重新定义初值
  TL0 = -50000%256;
  LED = ! LED;                        //电平取反
}
```

这个程序定义了一个发光二极管引脚 LED。在主程序中设置定时器寄存器参数，首先是开总允许中断 EA =1 和分允许中断 ET0 = 1（定时器 0 中断）。然后设置定时器工作方式 TMOD =0x01，即 T0 工作方式 1。接着是设置定时器 T0 的初值，高八位 TH0 和低八位 TL0。这里采用前面介绍的数字分离技术，因为 8 位 $2^8=256$，所以高 8 位用-5000/256 得到，低 8 位用-5000%256 得到。其中-5000 与 2^{16}-5000 结果是一样的。最后是启动定时器 0 即 TR0 = 1，停止定时器就是 TR0 =0。然后，主程序就进入一个无限循环。等待定时到，执行中断处理子程序。因此，主程序中没有什么动作，所有操作都是定时到后中断处理子程序进行的操作，即重新定义初值（因为工作方式 1 定时到后初值自动消失，需要重新定义），和改变发光二极管引脚的电平，使它闪烁。

下面我们就操作电路，编译执行这个程序。元器件列表见表 6-5。

表 6-5　元器件选择列表

序　号	元器件名称	名 称 说 明	备　注
1	LED-BLUE	发光二极管	蓝色
2	RES	电阻	阻值设定为 500Ω
3	89C52	单片机	51 系列

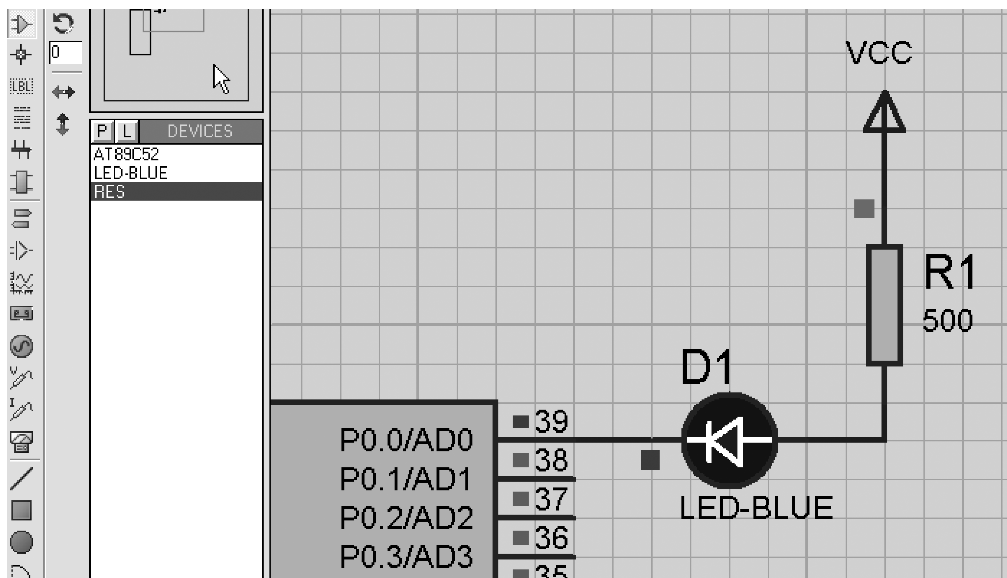


图 6-4　0.05s 闪烁一次

编译程序，在单片机上加载程序，可看到发光二极管 0.05s 的闪烁，如图 6-4 所示。由于间隔时间很短，发光二极管闪烁的很快。如何能使得间隔时间长一点呢？这就需要把定时器初值设的大一点，但是，最大也不能超过 2^{16}，即 65536，也就 65536μs = 0.065536s。如果要设置成 1s 闪烁一次该怎么办呢？

显然，定时器初值设置成 1s 是不可能的，只有另外设置变量来延长定时时间。我们修改一下中断处理程序如下：

```
/* 定时器 0 中断服务子程序，中断号 1   */
void intserv1 (void) interrupt 1 using 1
{
  TH0 = -50000/256;              //重新定义初值
  TL0 = -50000%256;
  t++;                           // 变量 t 加 1
  if (t>=20)                     //如果变量 t=20
  {
    t=0;                         //变量 t 清零
    LED=! LED;                   //电平取反
```

```
    }
}
```

在程序头部加上一个全局变量 t，定义为 unsigned char t;，作为中断子程序的中间变量。每次定时器时间到，处理中断时，给变量 t 加 1，若 t = 20，就是 0.05s × 20 = 1s。即每次 0.05s 时间到中断时并不执行发光二极管电平取反，直到 20 次时，就是 1s 时，才执行一次。这样就实现了定时 1s 的功能。重新编译程序，执行仿真运行，看是不是 1s 闪烁一次？

6.4　程序设计与仿真

以上我们学习了定时器的原理，并实现了 1s 定时功能。与外部中断相比，多了工作方式设置和定时器初值计算设置。定时/计数的中断编号是 1 和 3。

下面，我们就将定时计数功能用到数码管显示中来。其实，就是把自动加数的时间间隔改为 1s，就是 1s 数字加 1。这样，就需要在程序中加上定时器设置和中断处理子程序，原理图还是本章第 2 节的电路不变。我们在这个电路基础上显示时间的变化，前 2 位表示分钟，后 2 位表示秒。从 0 分 0 秒开始计时，逢 60 进 1，每秒变化一次。程序如下：

```
#include <reg51.h>                              //单片机头文件
unsigned char code Tab [ ] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
                              0x80, 0x90};      //共阳数码管码段表
unsigned char Dat [ ] = {0, 0, 0, 0};           //存放 4 位数字数组

int i, t;                                       //定义变量，作为循环，定时计数
unsigned char tmp;                              //定义片选变量
unsigned char Second =0, Min =0;                //定义秒、分变量
void Delay ( )                                  //延时子程序，作为数码管显示延时
{
  unsigned char i;
    for (i =0; i <250; i ++);
}

void main ( )
{
  EA =1;                                        //允许所有中断
  ET0 =1;                                       //允许 T0 中断
  TMOD =0x01;                                   //T0 方式 1 计时 0.05s
  TH0 =-50000/256;                              //定时器 T0 的高八位
  TL0 =-50000%256;                              //定时器 T0 的低八位
  TR0 =1;                                       //启动定时器 0
  while (1)                                     //无限循环
  {
```

```
        tmp =0x01;                              //片选初值
        for(i =0; i<4; i++)                     //循环 4 次
        {
            P2 = tmp;                           //片选赋值
          P0 = Tab [Dat [i]];                   //输出某一位数字的码段值
            tmp = tmp <<1;                      //片选值左移一位
            Delay ();                           //调用延时
        }
    }
}

/* 定时器 0 中断服务子程序 */
void intserv1(void) interrupt 1 using 1
{
    TH0 =-50000/256;                            //定时器高八位赋值
    TL0 =-50000%256;                            //定时器低八位赋值
    t++;                                        //变量 t 加 1
    if(t==20)                                   //20 次到，即 1 秒到
    {
        t=0;                                    //变量 t 清零
        Second++;                               //秒加 1
        if (Second> =60)                        //60 秒到
            {
                Second =0;                      //秒变量清零
                Min++;                          //分加 1
                if (Min> =60)                   //60 分到
                    Min =0;                     //分变量清零
            }
            Dat [0] =Min/10;                    //计算分高位
            Dat [1] =Min%10;                    //计算分低位
            Dat [2] =Second/10;                 //计算秒高位
            Dat [3] =Second%10;                 //计算秒低位
    }
}
```

这个程序的主程序在一开始设置了定时器的设置，在无限循环中只有显示 4 位数码管的工作，所以，程序很简洁，数码管的显示也没有抖动、闪烁现象。在中断处理子程序中，除了重新设置初值外，通过变量 t 确定是否到达 1s。到 t = 20 时，即 1s 时间到，先给秒位加 1，若秒位到达 60，则清零秒变量，并给分位加 1，同样，分位到达 60 也要清零。最后，分离 4 位数字，放入数组供输出，分、秒数码管显示如图 6-5 所示。

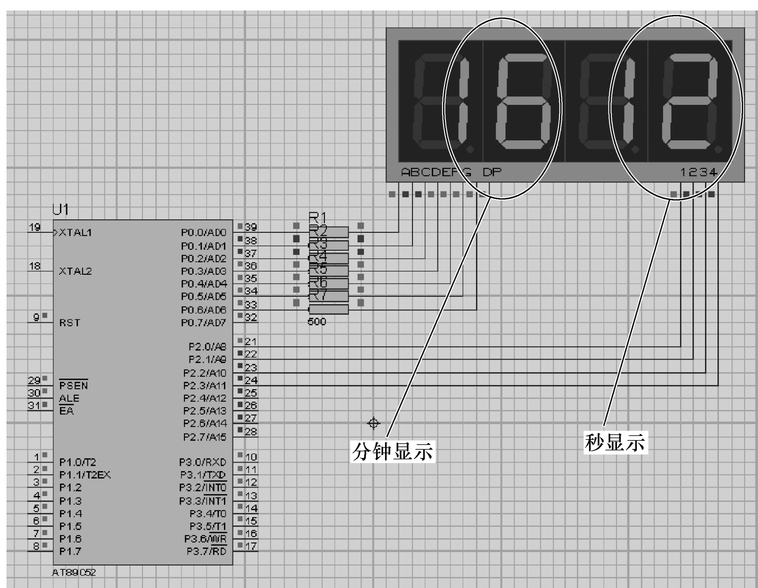


图 6-5　分、秒数码管显示

下面，我们在上述程序的基础上再增加 2 个按钮，用以设置分和秒的初值。利用前面外部中断的方法，当按钮 1 按下，就增加分的数。当按钮 2 按下，就增加秒的数。

按钮 1 接到 P3.2/INT0 引脚，按钮 2 接到 P3.2/INT1 引脚，另一端接地。因为，这两个引脚是系统规定的外部中断引脚外部中断 0 和外部中断 1。

在主程序中，增加外部中断的设置：

```
IT0 =1; EX0 =1;                          //开外部中断 0 和外部中断 0 允许
IT1 =1; EX1 =1;                          //开外部中断 1 和外部中断 1 允许
```

在程序后面加上 2 个外部中断的处理子程序：

```
void intersvr0 (void) interrupt 0 using 1      //外部中断 0 子程序
{
  Min ++;                                      //分加 1
  if (Min > =60)                               //如果分到 60
    Min =0;                                    //分清零
}

void intersvr1 (void) interrupt 2 using 1      //外部中断 1 子程序
{
  Second ++;                                   //秒加 1
  if (Second > =60)                            //如果分到 60
    Second =0;                                 //秒清零
}
```

完整的程序请读者自己完成，并进行编译、加载和仿真运行，图 6-6 所示为仿真显示用按钮调节分、秒数码管显示。

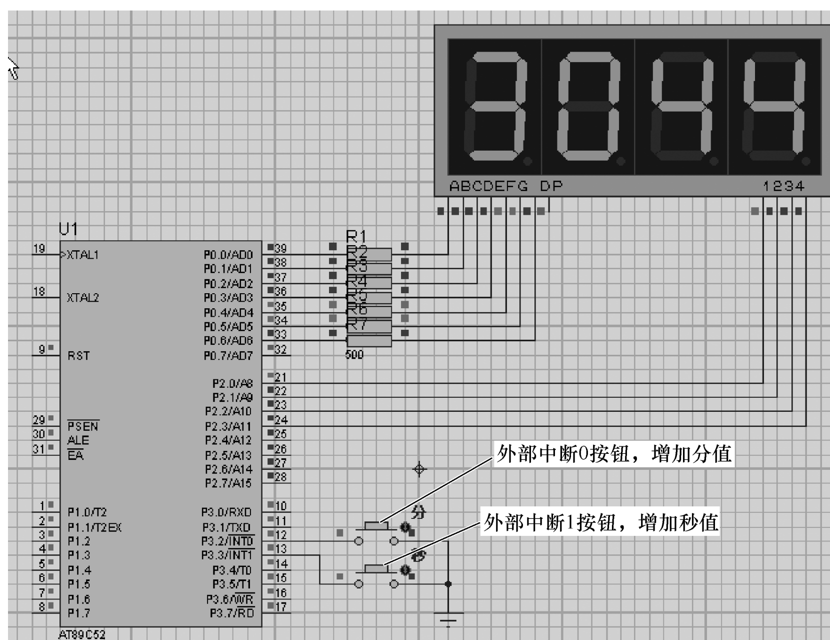


图 6-6　用按钮调节分、秒数码管显示

第7讲 蜂 鸣 器

导读

这一讲我们介绍蜂鸣器的应用。在单片机应用中，蜂鸣器使用是很普遍的，这是因为它价格低廉，经济实用，可作为一般的提示音的发声元件。在原理图设计中，为了使仿真运行更加生动有趣，我们将蜂鸣器 buzzer 用 speaker 代替，使得发声更洪亮，但是设计方法是一样的。我们将介绍蜂鸣器与单片机连接的电路设计和蜂鸣器的发声原理，以及编程技巧。

7.1 蜂鸣器应用电路及发声原理

首先我们看一下蜂鸣器与单片机连接的电路原理图，如图 7-1 所示。

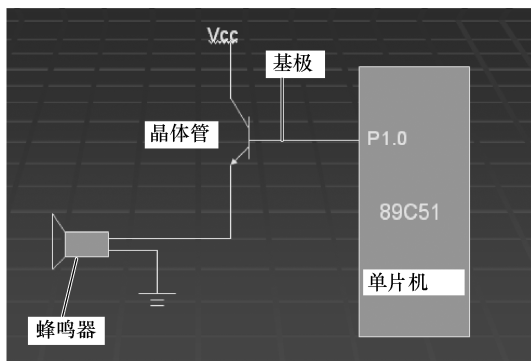


图 7-1 蜂鸣器控制原理图

我们看到，蜂鸣器与发光二极管不同，它不能直接连接到单片机引脚，因为它需要较大的电流驱动才能发声。这里在蜂鸣器的一端接一个晶体管，用单片机的引脚控制晶体管的基极，从而控制晶体管的导通和截止。根据晶体管的原理，基极电流很小，集电极到发射极的电流很大，这样，单片机就用小电流控制了大电流，最终控制蜂鸣器的开启和关闭。

接下去，我们来看蜂鸣器的发声原理。如果我们把蜂鸣器开启，是否就能发声了呢？还不能，我们只能听到“咔”的一声就没声音了。这是因为发声需要一定的导通和关闭的频率，即音频，频率高声音就尖，频率低声音就柔和。通过改变开启和关闭蜂鸣器的时间间隔，就改变了发声的频率，这在程序中很容易实现。

下面，我们就来设计蜂鸣器的应用电路，元器件列表见表 7-1。

表 7-1　元器件选择列表

序　号	元器件名称	名 称 说 明	备　注
1	SPEAKER（见图 7-2）	喇叭	替代蜂鸣器
2	NPN	标准 NPN 型晶体管	
3	RES	电阻	阻值设定为 200Ω
4	89C52	单片机	51 系列



图 7-2　选择蜂鸣器（喇叭）

图 7-3 所示为设计的原理图。

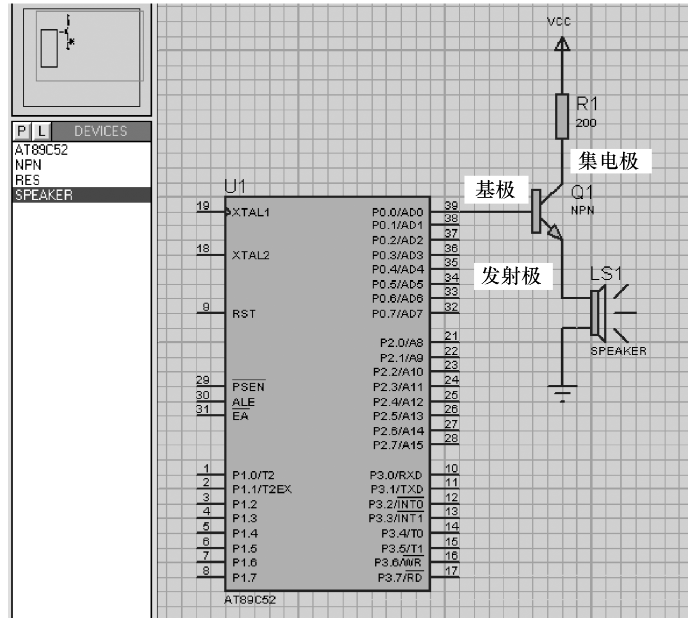


图 7-3　用晶体管驱动蜂鸣器控制电路设计原理图

将电阻阻值改为 200Ω，接电源，另一端接晶体管集电极。晶体管基极接单片机引脚 P0.0，发射极接蜂鸣器（SPEAKER），蜂鸣器另一端接地。这样，最简单的蜂鸣器应用电路

就设计好了。

7.2 程序设计与仿真

接着，我们来设计蜂鸣器发声的程序。首先，我们设计一个发出固定声音频率的程序。

```
#include <reg51.h>                    //单片机头文件
sbit BUZZER = P0^0;                   //定义蜂鸣器驱动引脚
void main (void)                      //主程序
{
    int i, n;                         //定义循环变量
    n = 60;                           //定义等待时间间隔为 60
    while (1)
    {
      BUZZER = 1;                     //蜂鸣器开
      for (i = 1; i < n; i ++);       //延时 60
      BUZZER = 0;                     //蜂鸣器关
      for (i = 1; i < n; i ++);       //延时 60
    }
}
```

从程序中我们看到，无限循环的工作就是打开蜂鸣器和关闭蜂鸣器，其中延时的时间间隔由变量 n 决定，n 的值大，延时就长，频率就低，反之频率就高。读者可以分别把 n 改为 200 和 20，然后编译、仿真运行，听一下蜂鸣器的发声频率发声的变化。

我们已经理解了蜂鸣器的发声原理，并知道如何控制发声频率。下面我们把上述程序修改成让蜂鸣器发出变化的声音，例如从高到低的变化，就像拉警报一样。根据前面的经验，只要能改变 n 的值，使它从小到大变化就行了。

```
#include <reg51.h>
sbit BUZZER = P0^0;
void main (void)
{
    int i, n;
    n = 0;                            //n 从 0 开始变化
    while (1)
    {
      BUZZER = 1;
      for (i = 1; i < n; i ++);
      BUZZER = 0;
      for (i = 1; i < n; i ++);
      n ++;                           //每循环一次加 1
      if(n > 1000) n = 0;             //直到 1000 后，n 清零
```

```
        }
}
```

这里只是把 n 变成了可变化的，也就是声音频率发生了变化。

我们再深入一下，把蜂鸣器发声变成子程序，只有在按下按钮时才发声。增加一个按钮放在 P3.2 引脚，如图 7-4 所示的方法选择按钮。



图 7-4　选择按钮

修改程序如下：

```
#include  <reg51. h >

sbit BUZZER = P0^0;
sbit BUTTON = P3^2;                    //定义按钮在引脚 P3. 2
void sound ( );                        //声明发声子程序

int i, n = 60;                         //定义循环变量和等待时间间隔

void main (void)
{
    while (1)
    {
      if (BUTTON == 0)                 //如果按钮按下
          sound ( );                   //调用发声子程序
    }
}

void sound ( )                         //发声子程序
{
    BUZZER = 1;
    for (i = 1; i < n; i ++ );
    BUZZER = 0;
    for (i = 1; i < n; i ++ );
}
```

这里我们把原来主程序中的蜂鸣器发声部分变成了子程序，因为放在最后，所以要在前面声明一下。变量在子程序中使用，因此，也不能放在主程序中，应放在前面作为全局变量了。现在主程序中只剩下一个无限循环，循环体就是判断按钮是否按下，若按下就调用蜂鸣器发声。图 7-5 所示为该电路仿真画面。

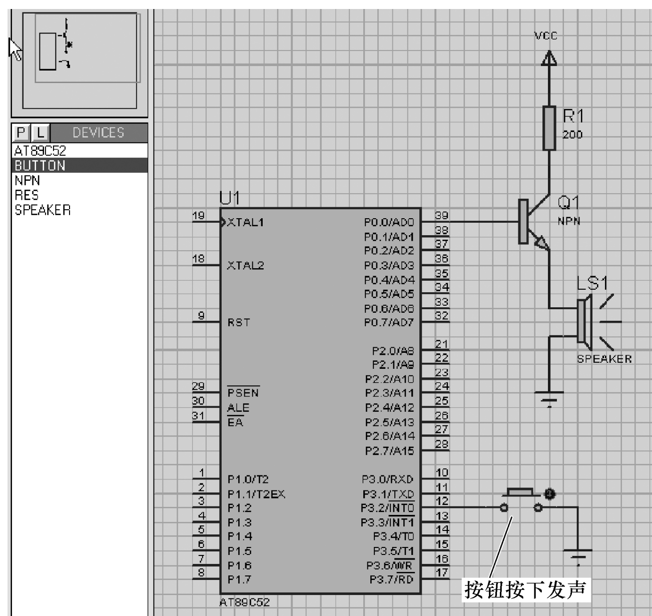


图 7-5 蜂鸣器控制电路仿真画面

编译程序，仿真运行。是不是像一个门铃？

第8讲 继　电　器

 导读

这一讲介绍单片机控制继电器的电路应用。继电器在工业控制中应用非常广泛，可以通过继电器对其他大电流的电器进行控制。

8.1 继电器驱动电路

首先看一下继电器控制的原理图，如图 8-1 所示。

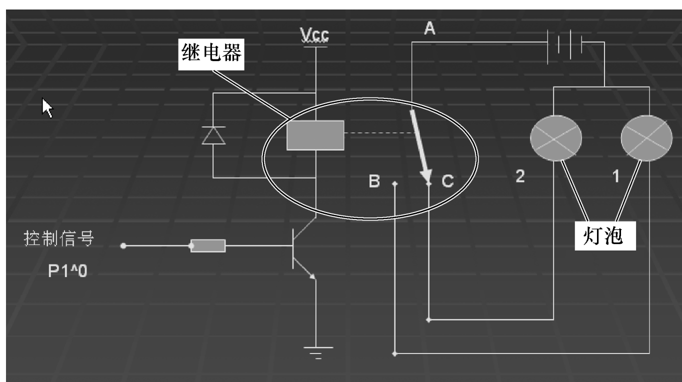


图 8-1　继电器控制原理图

继电器部分包括控制线圈和 3 个引脚，A 引脚接电源，B 是常开节点，C 是常闭节点。继电器没有通电时，常闭节点 C 构成通路，灯泡 2 点亮。继电器通电时，线圈吸合，常开节点 B 闭合，常闭节点 C 打开，灯泡 1 点亮，灯泡 2 关闭。

由于继电器通电的电流也很大，不能用单片机直接控制，需要用晶体管驱动，与控制蜂鸣器相同，用晶体管做开关电路，用单片机控制晶体管的基极控制晶体管的导通和截止，从而控制继电器的通电和关闭。

下面，我们来设计原理图，选择元器件列表见表 8-1。

为了搞清楚继电器的工作原理，我们先用继电器构成一个控制电路，用开关控制继电器，如图 8-2 所示。

表 8-1　元器件选择列表

序　　号	元器件名称	名称说明	备　　注
1	LAMP	灯泡	12V
2	BATTERY	电池	12V
3	RELAY	继电器	12V
4	SWITCH	开关	
5	NPN	标准 NPN 型晶体管	
6	RES	电阻	阻值设定为 500Ω
7	89C52	单片机	51 系列

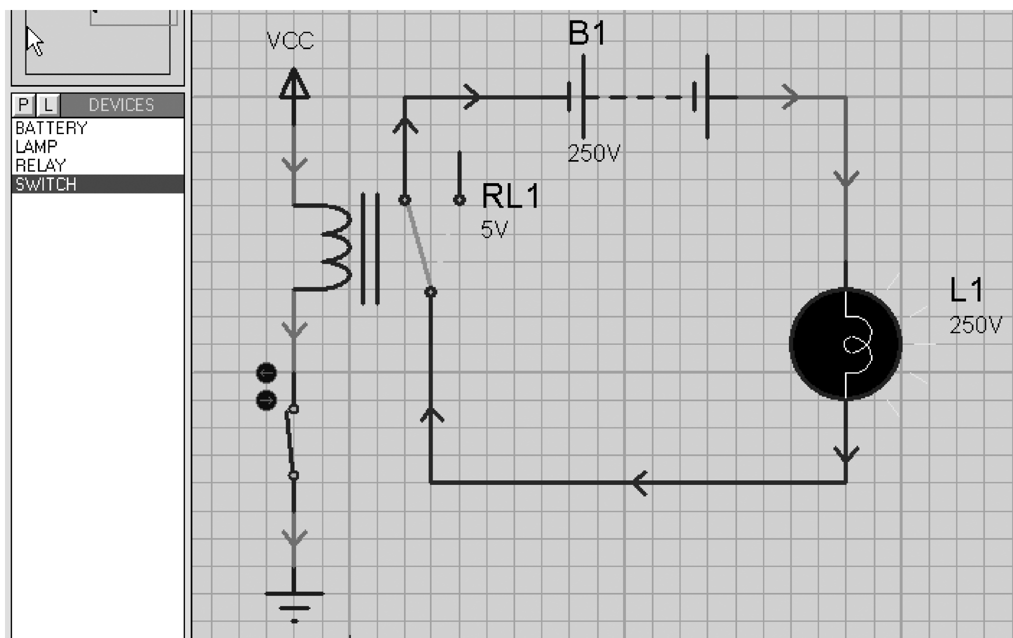


图 8-2　用开关控制继电器的电路

这里需要注意一个问题，即控制电器的电压值必须配对，就是说供电与电器的电压值必须一致，否则就会出问题，一是不能驱动（电压太低），二是电器烧毁（电压太高）。首先是继电器的电压值，原来是 12V 的，把它改成 5V，以符合 VCC 的电压值，如图 8-3 所示，若不改的话，我们就会发现，合上开关后，继电器不会吸合。

同样，电池和灯泡的电压值都改为 250V，如图 8-4 所示。

这样，就有两种电压的电路，继电器工作在 5V 电压，灯泡则工作在 250V 电压。开关控制继电器，继电器线圈吸合又控制了 250V 的电路导通，使灯泡点亮。

通过仿真运行这样一个例子，我们了解了继电器工作的基本原理，即用小电流控制大电流工作。下面，我们再接上单片机，用单片机控制继电器，用继电器控制灯泡。

在原来开关的位置，换上晶体管来控制继电器，如图 8-5 所示，再用单片机 P0.0 引脚控制晶体管的基极，从而控制的灯泡电路的导通与关闭。注意，接晶体管的电阻值要改为 500Ω，否则电流会太小，不能驱动晶体管导通。

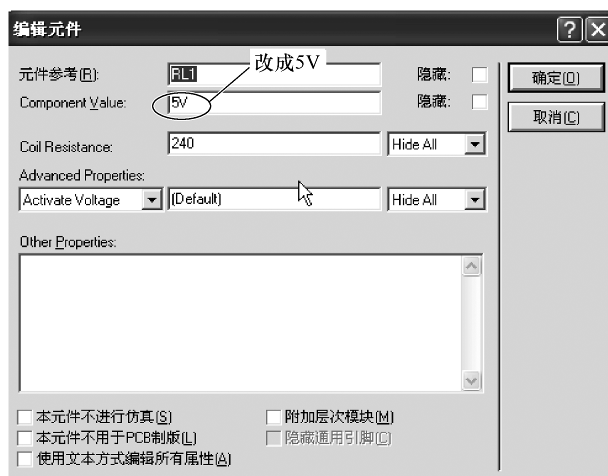


图 8-3　修改继电器的电压值

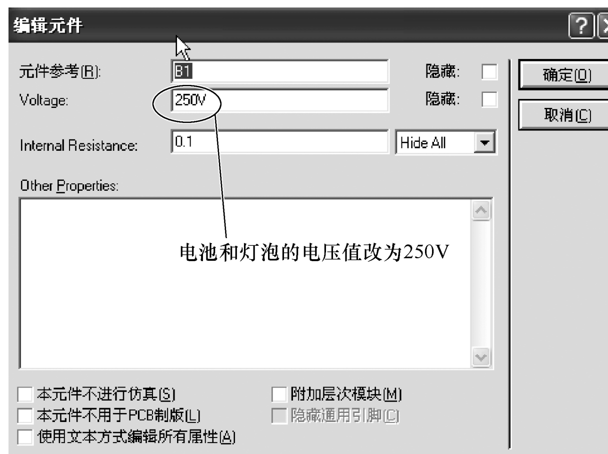


图 8-4　修改电池和灯泡的电压值

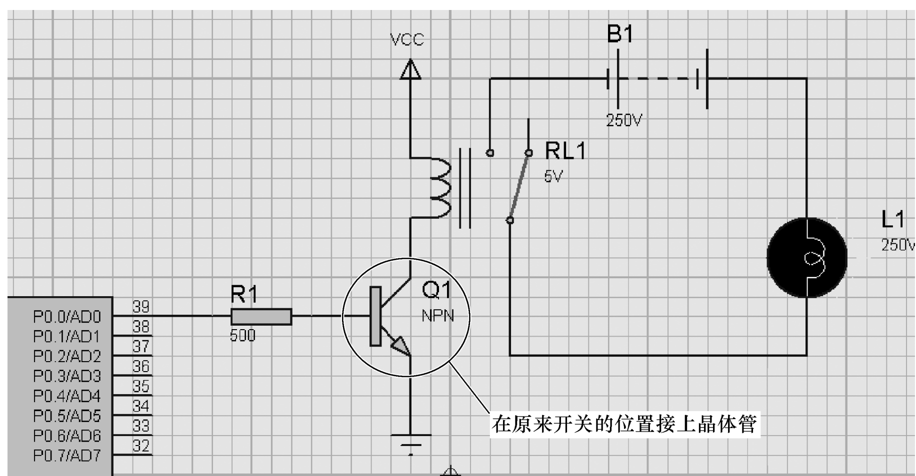


图 8-5　单片机引脚控制晶体管，用晶体管作为开关

最后，我们还要加上两个按钮，用以控制继电器的导通和截止。我们仍然在 P3.2 和 P3.3 引脚分别接一个按钮。

8.2　程序设计与仿真

接着，我们来设计继电器控制程序，程序如下：

```
#include <reg51.h>                  //单片机头文件
sbit S1 = P3^2;                     //定义按钮1，用于开灯
sbit S2 = P3^3;                     //定义按钮2，用于关灯
sbit RELAY = P0^0;                  //定义继电器引脚

void main ()                        //主程序
{
    while (1)                       //无线循环
    {
        if (S1 ==0) RELAY =1;       //如果按钮1按下，则继电器开通
        if (S2 ==0) RELAY =0;       //如果按钮2按下，则继电器关闭
    }
}
```

我们看到，继电器控制程序很简单。将继电器控制引脚定义到一个变量 RELAY，类型与按钮相同，也是 sbit，因为它也是 1 位。在主程序的无线循环中的内容，就是判断按钮是否按下，根据按钮的不同，打开或关闭继电器。将引脚赋值为 1，就是高电平，晶体管就会导通，继电器就打开了。反之，引脚赋值为 0，就是低电平，晶体管就截止，继电器就关闭了，控制电路如图 8-6 所示。

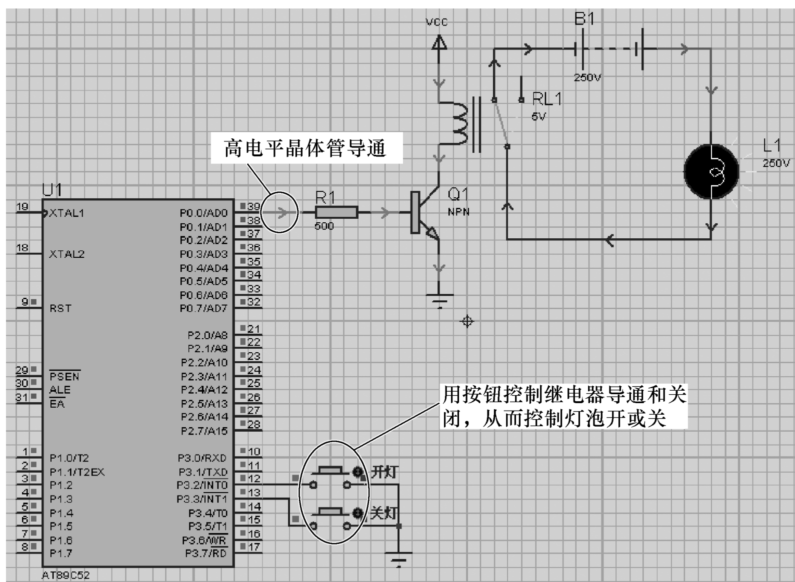


图 8-6　单片机继电器控制电路

可以看到，单片机编程都是针对硬件设置编写的，不同的硬件，不同的连接，就会有不同的程序。其次，需要根据单片机的功能，针对单片机编程，最终反映到单片机的引脚上，输出高电平或低电平，通过连接到引脚的器件控制其他电路。

为了在仿真时显示正负电压颜色和电流方向，可以设置菜单“系统”→“设置动画选项”，在对话框中勾选“Show Wire Voltage by Colour?”和”Show Wire Current with Arrows?”，如图 8-7 所示。

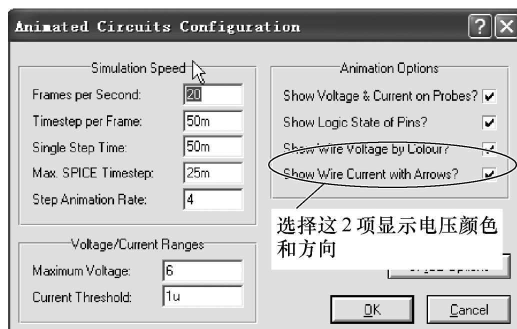


图 8-7　设置动画选项

第 9 讲

定时供电插座设计

 导读

前面我们介绍了单片机控制一些常用外部器件的电路及程序设计，每一个都是单独与单片机相连的。下面我们用这些器件来设计一个实用的产品，做一个“家用定时供电插座”，这就需要涉及各种器件——继电器、数码管、按键、发光二极管、蜂鸣器等——的综合应用。

9.1 定时供电插座需求分析

设计一个产品，一般需要以下几步：

1）产品需求分析；

2）产品硬件分析与设计；

3）程序设计与调试。

当然，真正的产品设计还要进一步细化、分工合作完成。若产品比较简单，也可以从头至尾由一人完成。

首先，我们进行“家用定时供电定时插座”的需求分析。什么产品是需求分析呢？就是调查研究这个产品要实现什么功能，是否有市场，若需要这些功能，在技术上能否实现。所谓“家用供电定时插座”，就是提供一个供电插座，就像我们家里或单位里用的那种接线板，将电源接到一个插座板中，供多个电器使用。不同的是，这个插座板的供电时间可以设定，即可以让它定时供电或断电。这样，我们就可以用它间接地控制插在它上面的电器了。如电热水器、电瓶车充电器，让它在用电低谷供电，高峰断电，以节省用电费用。

从上面的分析看，市场上还是很需要这种功能的产品的。并且，这种产品也不复杂，就是普通的插座加上定时功能即可，定时功能可以用控制继电器的通、断来实现，用单片机实现是很方便的。为了设置定时，还需要有时间显示器件，这可以用数码管实现，再加上几个按钮就成了。

通过以上分析，我们得出了产品的基本结构：

1）有一个或多个受控继电器；

2）有数码管显示定时供电或断电时间；

3）有按钮可设置时间。

进一步的需求还要了解和确定继电器的功率大小，电压范围等，以保证用电器的正常使用。对于单片机可控制的继电器，最好使用控制电压 5V 的继电器，这样电路就比较简单。接触点电压应该符合电器使用的电压范围，如 220V。另外，还要考虑用电安全的问题，防止强电与弱电串扰，引起触电事故。因此，要考虑强电与弱电隔离的方法和措施。这里我们就不在深入讨论了，望有兴趣的读者查找其他书籍资料进行研究学习。

数码管显示时间精确到分就可以了，因此，4 位数码管就够了，2 位显示小时，2 位显示分。

按钮至少要考虑控制小时的设置、分的设置，还可以考虑启动和停止按钮。所以，至少 2 个。

总之，元器件多一点，控制就会方便一些，但是成本也就上去了，这就需要我们综合考虑，找到一个合理的平衡点。

9.2 定时供电插座电路设计

下面我们就来进行定时供电插座的硬件和电路设计。根据以上的分析，在电路中我们需要加入一个 4 位的数码管，2 个按钮，一个继电器，再加一个蜂鸣器作为时间到的提示。可以把前面的案例进行组合，并进行适当的调整就可以了。

选择元器件列表见表 9-1。

表 9-1 元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	7SEG-MPX4CA	4 位数码管	
2	BUTTON	按钮	
3	RELAY	继电器	5V/220V
4	SPEAKER	蜂鸣器	5V
5	NPN	标准 NPN 型晶体管	
6	RES	电阻	阻值设定为 500Ω
7	89C52	单片机	51 系列

根据以上列表，我们来设计电路，这个电路如图 9-1 所示。

为了加快设计速度，我们完全可以从原有的电路图复制所需要的部分，只要改动部分单片机引脚就行了。

首先，可以确定一个原有的电路作为基础，在上面添加所需的部分。这里我们以继电器控制电路作为基础。将原有的电路设计图文件复制到一个新的设计图，打开新设计图，如图 9-2 所示。

为了在 P0 和 P2 口放置 4 位数码管，需要将继电器部分移动至左边。断开单片机 P0.0

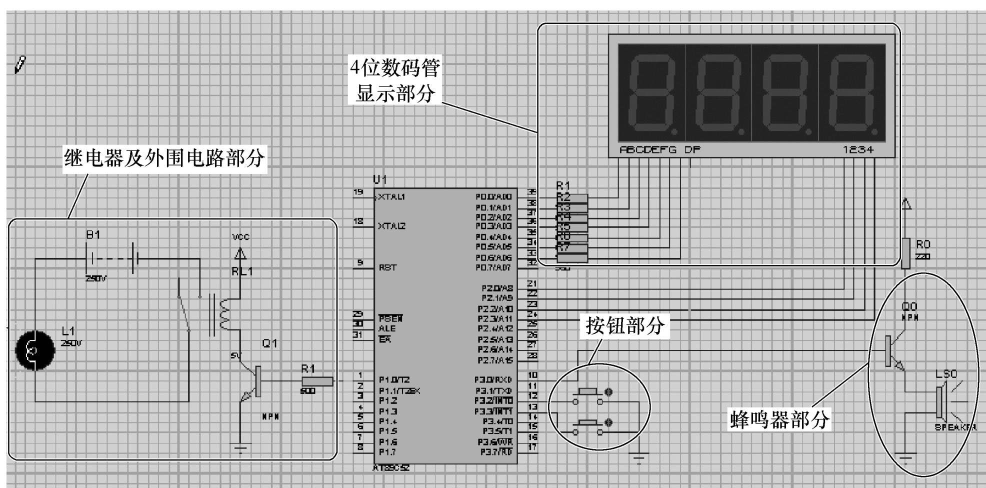


图 9-1　定时供电插座电路

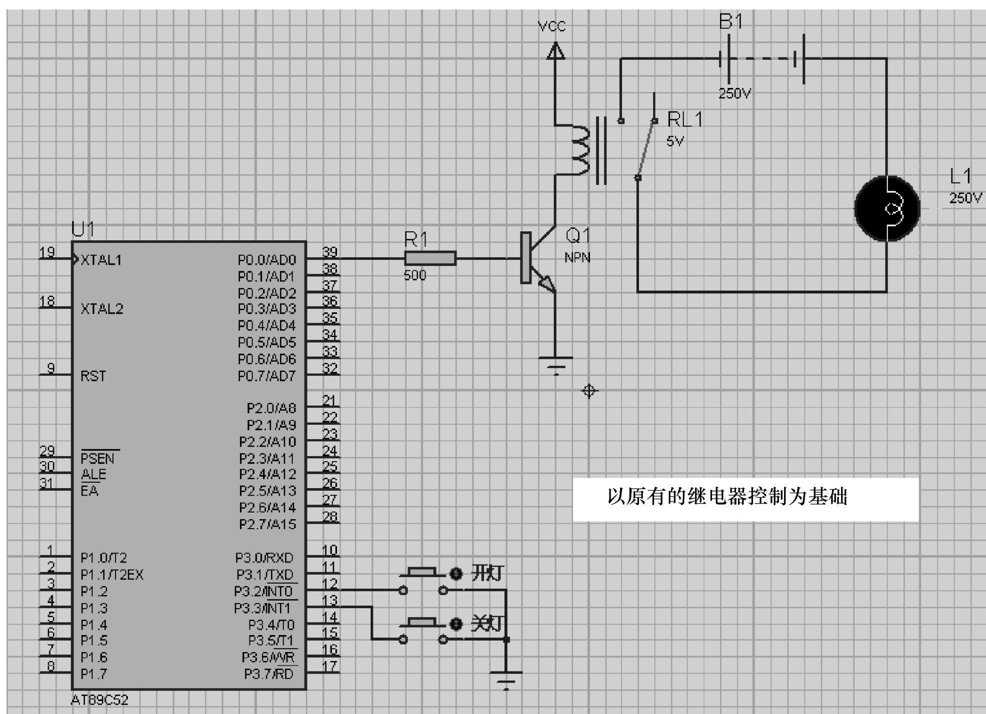


图 9-2　原有继电器控制电路作为基础

与电阻的引脚，选中继电器部分电路，如图 9-3 所示，将其移动至左边，并水平镜像翻转一下，如图 9-4 所示，使得电阻能够方便与单片机左边引脚相连。

水平翻转以后，就可以很方便地将电阻引脚连接到单片机 P1.0 引脚了。接着，我们要把蜂鸣器部分从现成的电路中复制过来，通过向文字复制粘贴一样的方法从另一个电路复制到这个电路中来。首先打开蜂鸣器电路图，选中需要复制的部分，单击鼠标右键，在弹出的菜单中选择“复制”，如图 9-5 所示。

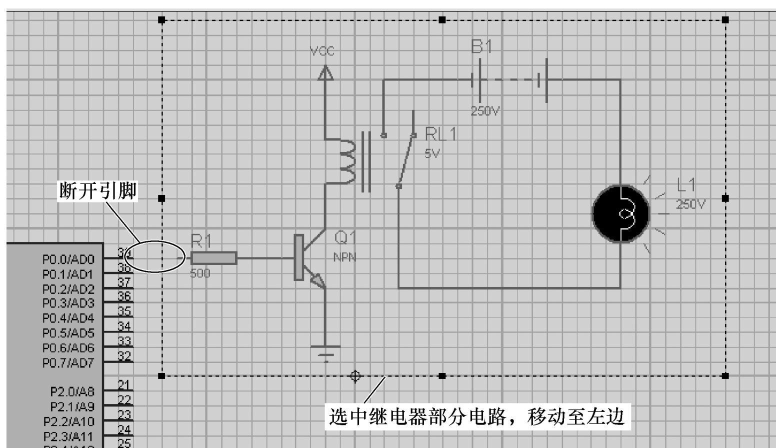


图 9-3　将继电器部分移动至左边

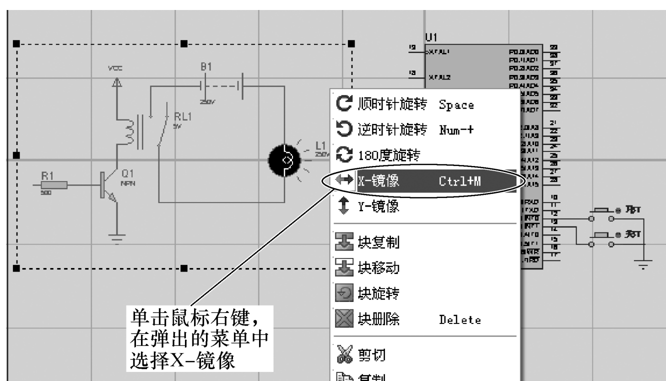


图 9-4　将电路水平镜像翻转

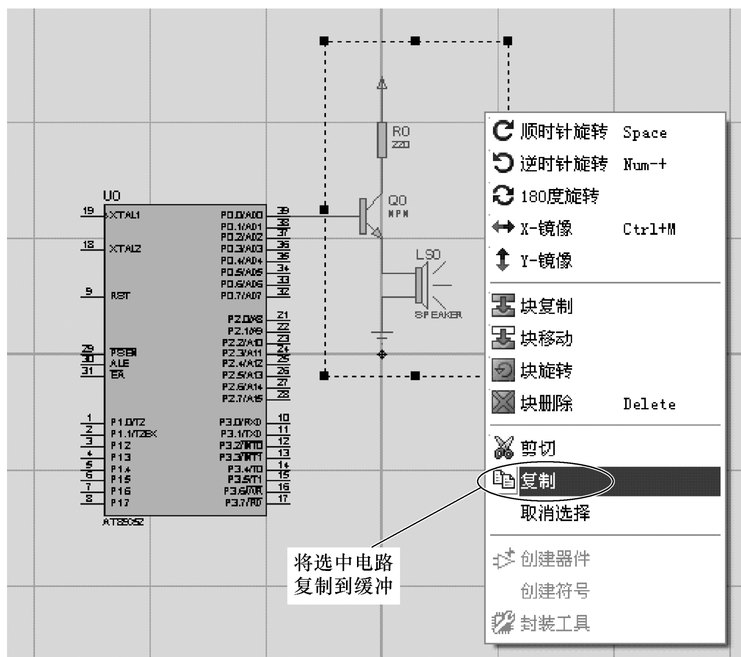


图 9-5　复制蜂鸣器部分电路

然后，在电路中粘贴缓冲区中的电路到编辑区，如图9-6所示，粘贴后的效果如图9-7所示。

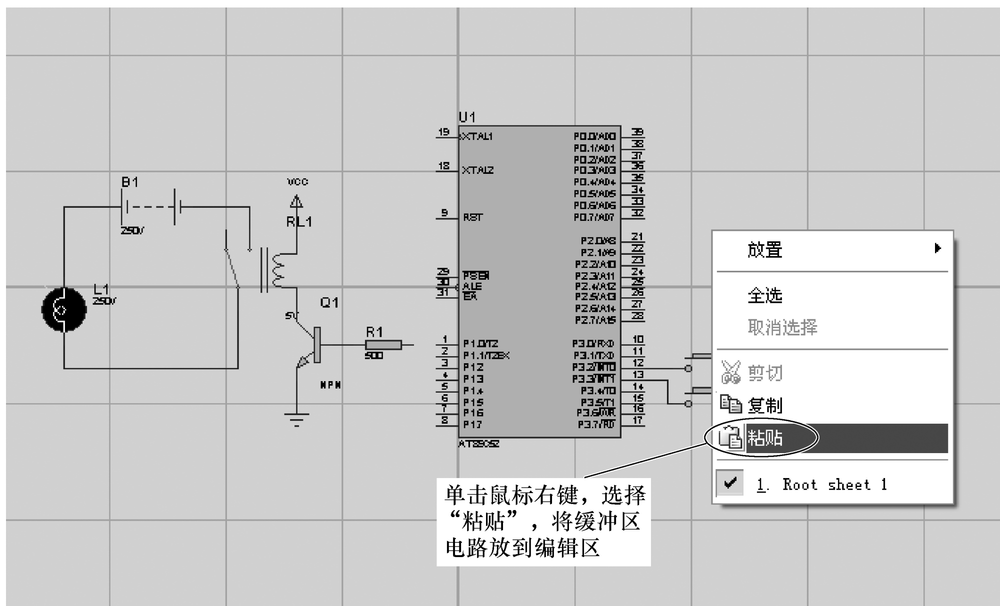


图9-6　粘贴缓冲区中的电路到编辑区

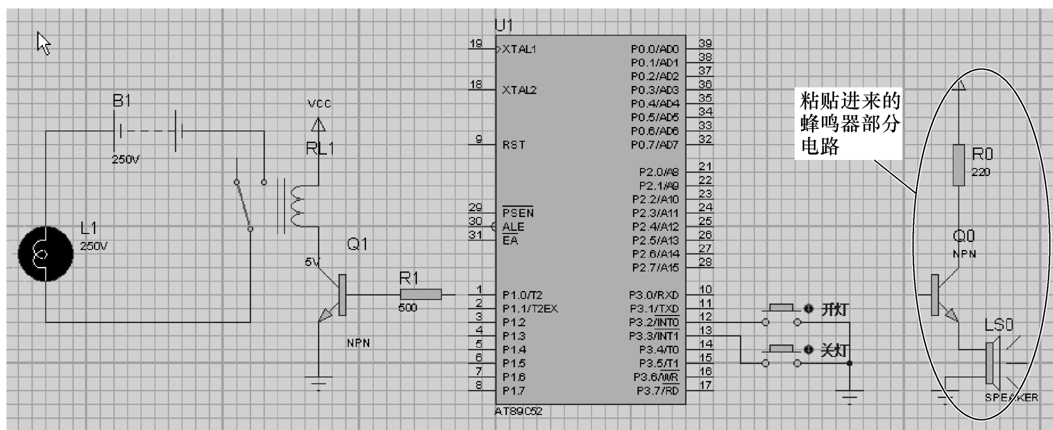


图9-7　蜂鸣器部分电路粘贴进来

然后，我们还要复制数码管部分电路到编辑区。这次，我们换一种电路导入的方法，用文件作为过渡。打开4位数码管电路图，选择需要复制的数码管部分，如图9-8和图9-9所示。

回到原理图设计，选择菜单“导入区域”，从保存的文件导入数码管部分电路，如图9-10所示。

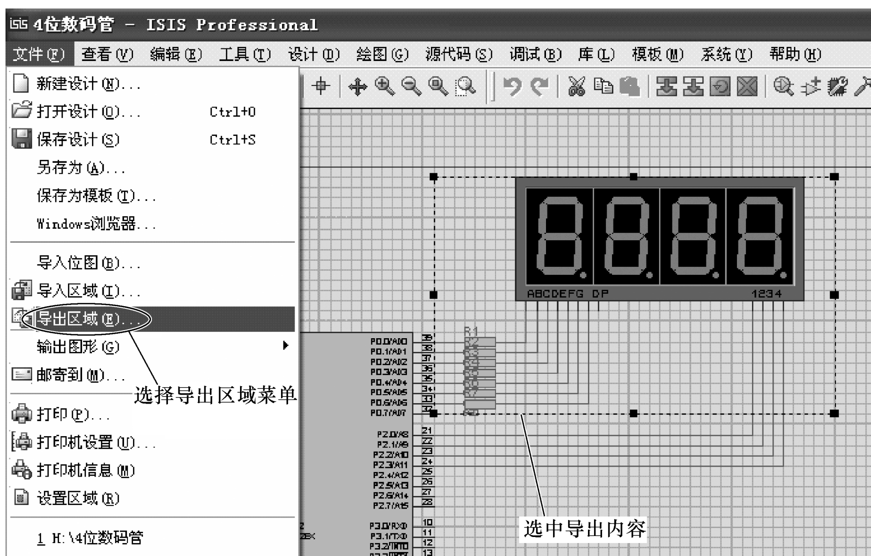


图 9-8　选择导出区域菜单

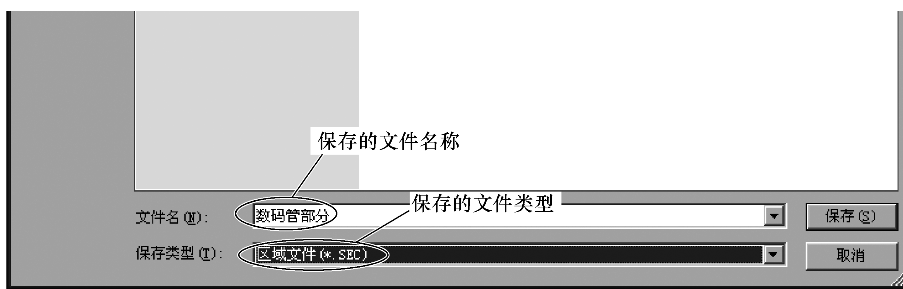


图 9-9　保存区域文件

选好导入文件后，在编辑区会出现一个框，显示导入电路所需的大小区域，让你选择导入位置，如图 9-11 所示。

这样，通过调整和连线，整个电路就完成了，如图 9-12 所示。

完成后的端口连接如下：

4 位数码管数据线	P0
数码管位控制线	P2.0 ~ P2.3
蜂鸣器控制	P3.0
继电器控制	P1.0
按钮控制	P3.2，P3.3

以上，我们利用原有的设计，通过移动、复制或区域导入，快速地完成了目标电路的原理图设计。通过复制或区域导入后，我们可以发现，元件列表窗口自动地加入了原理图中新添加的元器件。

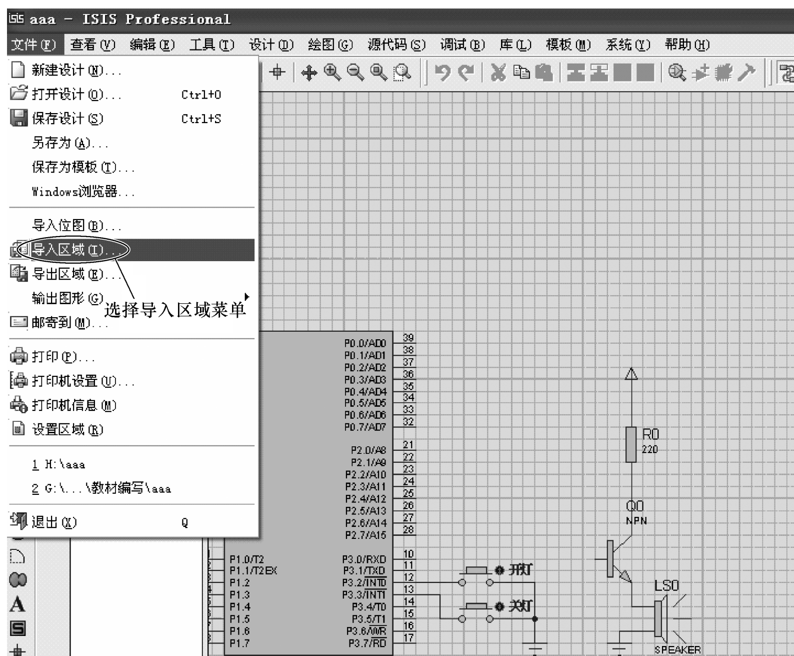


图 9-10　在编辑区选择导入区域

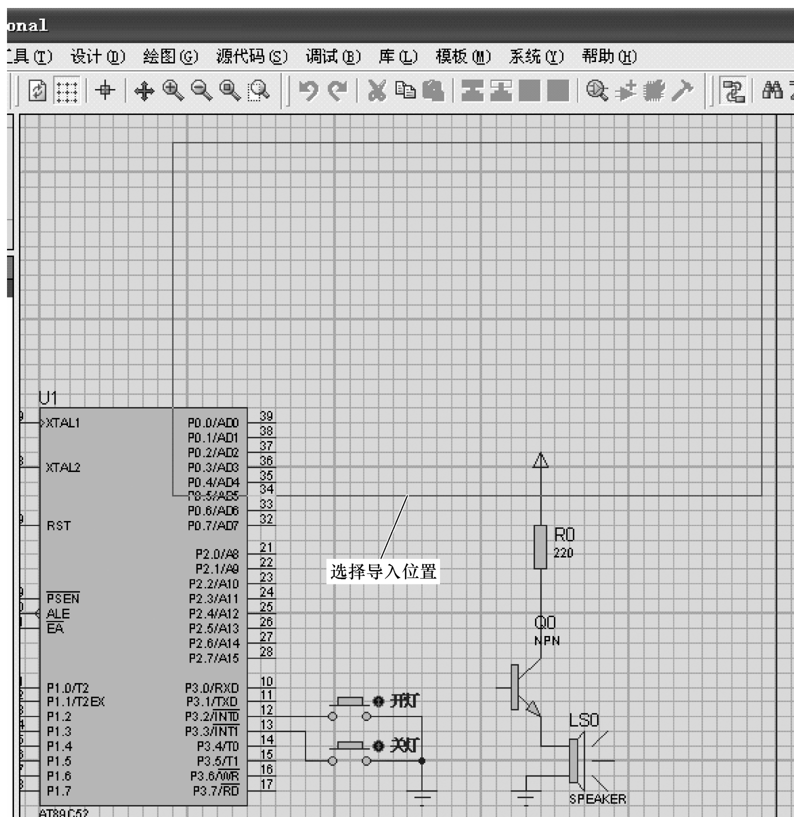


图 9-11　选择导入位置

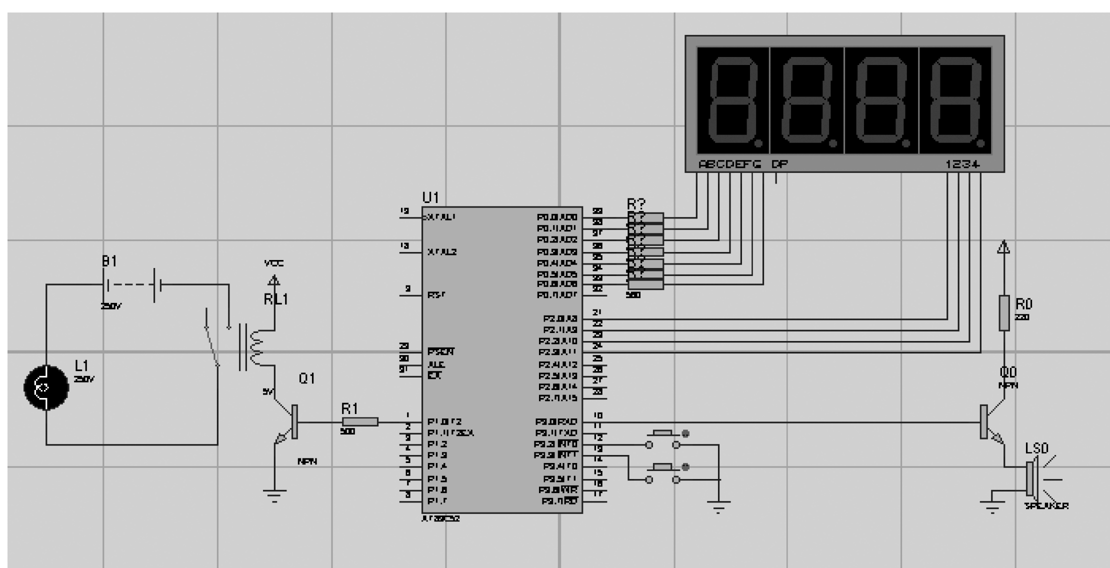


图 9-12　完成的设计图

注意：通过复制电路完成的设计，可能会有元器件编号重复的现象出现，需要及时纠正，否则将不能进行仿真，出现类似如图 9-13 所示的提示：

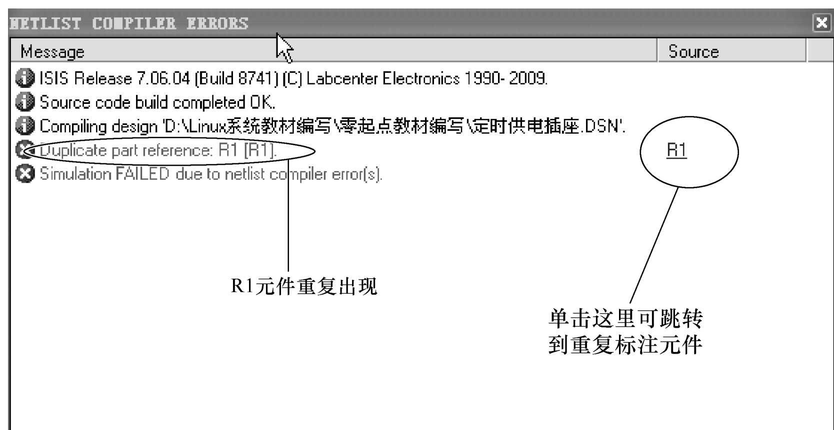


图 9-13　重复标注元件编号错误

这里，我们发现有两个 R1 编号的电阻元件，要把其中一个改为新的编号，如 R9。

9.3　程序设计与仿真

下面我们进行定时供电插座的程序设计。前面已经分析了定时供电插座的基本功能，根据这些功能，就可以确定程序的框架。程序流程如图 9-14 所示。

为了使仿真能比较快的看到结果，这里把小时、分改为分、秒，读者在实际使用中只要稍作更改就可以了。

根据流程图可以看到，程序的循环过程就是设置时间、倒计时、判断时间，其中倒计时显示时间又是一个子程序。在前面的时间显示程序中我们知道，时间处理可以通过单片机的定时器中断处理子程序来实现。

因此，可以用定时中断处理子程序实现时间的更新（倒计时）、判断时间是否到（时间为 0），若时间到就执行供电或断电即可。这样，主循环只负责时间设置（若有设置请求）、时间显示就行了。

这里的时间设置请求也可以用前面讲到的外部中断请求实现，把处理程序变成中断处理子程序。所以，主循环就只剩下时间显示了。

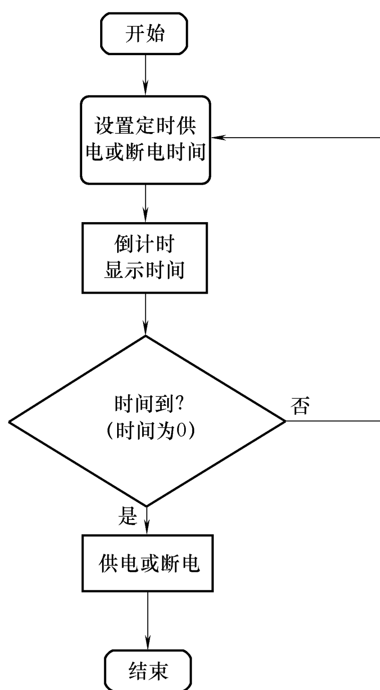


图 9-14　程序流程图

最后，我们讨论一下供电和断电问题。如果你对继电器比较了解，这个问题就很好解决了。我们在继电器原理图讲到，继电器有常开节点和常闭节点，还有一个公共节点。当继电器未通电时，即没有导通时，常开节点是断开的，常闭节点是闭合的。反之，当继电器通电时，常开节点闭合，常闭节点断开。因此，若你的插座接在常开节点，则一开始是断开的，时间到则闭合通电。反之，插座接在常闭节点，一开始就是通电的，时间到就会断开断电。所以，一个继电器的两个节点可以分别接两个插座，标上定时供电和定时断电即可。我们这个电路中的一个节点是空着的，你可以再接一路电器就显示出来了。当然，你也可以把程序做成时间到后供电，延续一段时间后再断电，这个问题留给读者自己去做。

下面我们就开始编程。与硬件设计类似，同样可以把以前设计的代码复制过来用，提高设计速度。首先把 4 位数码管时间显示的代码复制进来。因为端口没有变，因此，编译后直接就可以运行了。

```
#include <reg51.h>                           //单片机头文件
unsigned char code Tab[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                             //共阳数码管码段表
unsigned char Dat[] = {0,0,0,0};             //存放 4 位数字数组

int i,t;                                     //定义变量,作为循环,定时计数
unsigned char tmp;                           //定义片选变量
unsigned char Second = 0,Min = 0;            //定义秒、分变量
void Delay()                                 //延时子程序,作为数码管显示延时
{
  unsigned char i;
    for(i = 0;i < 250;i ++);
}
```

```
void main( )
{
   EA  = 1;                              //允许所有中断
   ET0  = 1;                             //允许 T0 中断
   TMOD  =0x01;                          //T0 方式 1 计时 0.05s
   TH0  = -50000/256;                    //定时器 T0 的高八位
   TL0  = -50000%256;                    //定时器 T0 的低八位
   TR0  = 1;                             //启动定时器 0
   while(1)                              //无限循环
   {
      tmp =0x01;                         //片选初值
      for(i =0;i <4;i ++ )               //循环 4 次
      {
            P2 = tmp;                    //片选赋值
         P0 = Tab[ Dat[ i] ];            //输出某一位数字的码段值
            tmp = tmp << 1;              //片选值左移一位
            Delay( );                    //调用延时
      }
   }
}

/* 定时器 0 中断服务子程序 */
void intserv1 ( void) interrupt 1 using 1
{
   TH0 = -50000/256;                     //定时器高八位赋值
   TL0 = -50000%256;                     //定时器低八位赋值
   t ++ ;                                //变量 t 加 1
   if( t ==20)                           //20 次到,即 1 秒到
   {
      t =0;                              //变量 t 清零
      Second ++ ;                        //秒加 1
      if( Second >=60)                   //60 秒到
            {
                  Second =0;             //秒变量清零
                  Min ++ ;               //分加 1
                  if( Min >=60)          //60 分到
                     Min =0;             //分变量清零
            }
```

```
            Dat[0] = Min/10;              //计算分高位
            Dat[1] = Min%10;              //计算分低位
            Dat[2] = Second/10;           //计算秒高位
            Dat[3] = Second%10;           //计算秒低位
    }
}
```

这样，我们就已经解决了时间显示问题。下面我们来解决时间设置问题，这里要用到外部中断处理程序，就是第 5 讲的按钮处理，把两个按钮处理的外部中断设置和中断处理子程序复制进来。

外部中断设置：

```
IT0 = 1;EX0 = 1;                    //开外部中断 0 和外部中断 0 允许分开关
IT1 = 1;EX1 = 1;                    //开外部中断 1 和外部中断 1 允许分开关
```

外部中断子程序：

```
void intersvr0(void)interrupt 0 using 1         //外部中断 0 处理子程序
{
  p ++ ;
  if(p >9)
    p =9;
}

void intersvr1(void)interrupt 2 using 1         //外部中断 1 处理子程序
{
  p-- ;
  if(p <1)
    p =1;
}
```

把它们改成对分和秒进行处理，这里外部中断 0 对分处理，外部中断 1 对秒处理。因为只有一个按钮，就只能加，加到头清零。这里分、秒都是 60 进制，所以到 60 就清零。

```
void intersvr0(void)interrupt 0 using 1         //外部中断 0 处理子程序
{
  Min ++ ;                                      //分加 1
  if(Min >=60)                                  //若大于等于 60
    Min =0;                                     //分清零
}

void intersvr1(void)interrupt 2 using 1         //外部中断 1 处理子程序
{
  Second ++ ;                                   //秒加 1
  if(Second >=60)                               //若大于等于 60
```

```
        Second = 0;                         //秒清零
    }
```

将中断设置加入主程序头部，中断子程序放在程序尾部。这样，我们已经完成了时间设置问题。接着，我们要解决时间倒计时问题。原来显示时间是往上加 1，现在改为减 1，其他设置相应改动即可，这里只要把定时中断处理子程序改一下：

```
    void intserv2(void)interrupt 1 using 1
    {
      TH0 = -50000/256;                //定时器高八位赋值
      TL0 = -50000%256;                //定时器低八位赋值
      t ++;                            //变量 t 加 1
      if(t ==20)                       //20 次到,即 1 秒到
      {
        t =0;                          //变量 t 清零
        Second--;                      //秒减 1
        if(Second <0)                  //秒减到负值
            {
                Second =59;            //秒变量设为 59
                Min--;                 //分减 1
                if(Min < =0)           //分减到 0
                  Min =0;              //分最小为零,不能为负值
            }
            Dat[0] = Min/10;           //计算分高位
            Dat[1] = Min%10;           //计算分低位
            Dat[2] =Second/10;         //计算秒高位
            Dat[3] =Second%10;         //计算秒低位
      }
    }
```

以上秒到负值时要变成 59，但分不能变为最大值。为了使初始不至于就停止，可以给分、秒赋一个初值，如 3∶30。

```
    unsigned char Second =30,Min =3;            //定义秒、分变量
```

接着，我们要解决最重要的问题，就是时间到，要开继电器。首先要在开始定义继电器引脚，现在，继电器控制端接在 P1.0，定义如下：

```
    sbit  JDQ = P1^0;                          //定义继电器控制端引脚
    JDQ =0;                                    //开始为关闭状态,放在主程序开头
```

然后，要在定时中断程序中，根据条件开启继电器。这个条件就是，时间为 0，即分和秒都是 0：

```
    if(Min ==0 && Second ==1)                 //(为了稳定起见,我们设在最后 1 秒停止)
    {
        JDQ =1;                               //开继电器
```

```
    TR0 = 0;                            //关定时器
}
```

剩下一个问题就是蜂鸣器发声，我们让它在定时到后发声一段时间（长时间不断发声会有扰民问题呀）。与继电器一样，先要定义蜂鸣器控制端引脚，现在蜂鸣器接在 P3.0，定义如下：

```
sbit  BUZZER  = P3^0;                   //定义蜂鸣器控制端引脚
```

我们把第 7 讲蜂鸣器发声子程序复制过来稍作改动，加上一个循环，以便延时一段时间：

```
void sound()                            //发声子程序
{
    int i,n = 200;
   for(i = 0;i < 250;i ++)              //延时一段时间
    {
        BUZZER = 1;
        for(i = 1;i < n;i ++);
        BUZZER = 0;
        for(i = 1;i < n;i ++);
    }
}
```

把发声子程序放在程序最后。由于子程序放在最后，因此需要在开始声明一下：

```
void sound();                           //声明发声子程序
```

在时间到判断中，加上发声子程序调用：

```
if(Min == 0 && Second == 1)             //(为了稳定起见,我们设在最后 1 秒停止)
{
    JDQ = 1;                            //开继电器
    TR0 = 0;                            //关定时器
    sound();                            //调用发声子程序
}
```

到这里，我们的程序就大功告成了。以下是刚才讨论的程序内容：

```
#include  <reg51.h>                     //单片机头文件
unsigned char code Tab[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                        //共阳数码管码段表
unsigned char Dat[] = {0,0,0,0};        //存放 4 位数字数组
sbit  JDQ = P1^0;                       //定义继电器控制端引脚
sbit  BUZZER = P3^0;                    //定义蜂鸣器控制端引脚
void sound();                           //声明发声子程序
int i,t;                                //定义变量,作为循环,定时计数
unsigned char tmp;                      //定义片选变量
unsigned char Second = 30,Min = 3;      //定义秒、分变量
```

```
void Delay( )                          //延时子程序,作为数码管显示延时
{
  unsigned char i;
    for(i =0;i <250;i ++ );
}

void main( )
{
  JDQ =0;                              //开始为关闭状态
  EA  =  1;                            //允许所有中断
  IT0 =1;EX0 =1;                       //开外部中断 0 和外部中断 0 允许分开关
  IT1 =1;EX1 =1;                       //开外部中断 1 和外部中断 1 允许分开关

  ET0  =  1;                           //允许 T0 中断
  TMOD  =0x01;                         //T0 方式 1 计时 0.05s
  TH0  =  -50000/256;                  //定时器 T0 的高八位
  TL0  =  -50000%256;                  //定时器 T0 的低八位
  TR0  =  1;                           //启动定时器 0
  while(1)                             //无限循环
  {
    tmp =0x01;                         //片选初值
    for(i =0;i <4;i ++ )               //循环 4 次
    {
      P2 = tmp;                        //片选赋值
      P0 = Tab[Dat[i]];                //输出某一位数字的码段值
      tmp = tmp << 1;                  //片选值左移一位
      Delay( );                        //调用延时
    }
  }
}

/* 定时器 0 中断服务子程序 */
void intserv2(void)interrupt 1 using 1
{
  TH0 = -50000/256;                    //定时器高八位赋值
  TL0 = -50000%256;                    //定时器低八位赋值
  t ++ ;                               //变量 t 加 1
  if(t ==20)                           //20 次到,即 1 秒到
  {
```

```
        t = 0;                                  //变量 t 清零
        Second--;                               //秒减 1
        if(Second < =0)                         //秒减到 0
        {
            Second = 59;                        //秒变量设为 59
            Min--;                              //分减 1
            if(Min <= 0)                        //分减到 0
              Min = 0;                          //分最小为零,不能为负值
        }
        Dat[0] = Min/10;                        //计算分高位
        Dat[1] = Min%10;                        //计算分低位
        Dat[2] = Second/10;                     //计算秒高位
        Dat[3] = Second%10;                     //计算秒低位
          if(Min == 0 && Second == 1)           //分为 0、秒为 1 时
            {
                JDQ = 1;                        //开继电器
                TR0 = 0;                        //关定时器
                sound();                        //调用发声子程序
            }

    }
}

void intersvr0(void)interrupt 0 using 1        //外部中断 0 处理子程序
{
    Min ++ ;
    if(Min  >=60)
      Min  =0;
}

void intersvr1(void)interrupt 2 using 1        //外部中断 1 处理子程序
{
    Second ++ ;
    if(Second  >=60)
      Second  =0;
}

void sound()                                    //发声子程序
{
```

```
    int i,n = 200;
    for(i = 0;i < 250;i ++ )                    //延时一段时间
    {
        BUZZER = 1;
        for(i = 1;i < n;i ++ );
        BUZZER = 0;
        for(i = 1;i < n;i ++ );
    }
}
```

最后，我们在继电器常闭节点也接上灯泡，一开始常闭节点灯亮，常开节点灯不亮，时间到后反之，仿真运行结果如图 9-15 所示。

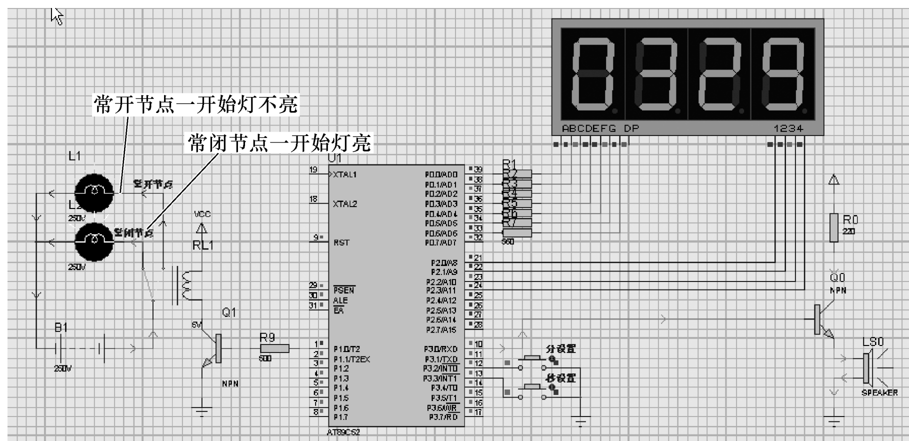


图 9-15　定时供电插座仿真运行

第 10 讲

RS232 串口通信

 导读

计算机与其他设备通信，就需要有一种方式，如串行接口（串口）、并行接口（并口）等。在单片机中内置了串口通信方式，通过规定的引脚 RXD（P3.0）和 TXD（P3.1）进行通信。本讲我们就来学习如何通过串口方式在单片机与单片机、单片机与 PC 机、多个单片机之间进行通信。

首先，简单介绍一下通信的基本理论。

■ 通信方式

（1）并行通信

1）通信一次同时可以传送一个数据字节（如 8 位）；

2）优点：传输速度快；

3）缺点：需要连接线数多，传输成本高。

（2）串行通信

1）只能按字节逐位传输（每次传送 1 位）；

2）优点：线路少，线路成本低；

3）缺点：传输速度慢。

（3）异步通信

1）通常以字符为单位组成字符帧传送；

2）由起始位、数据位、奇偶校验位和停止位组成。

（4）同步通信

1）发送和接收时时钟保持严格同步；

2）一大批数据分成几个数据块，数据块之间与同步字符隔开，而各位二进制之间没有间隔。

（5）波特率

1）定义：每秒传输的二进制位，如 9600bit/s；

2）通信双方必须有相同的波特率。

上述基本概念是我们进行计算机通信的基础，下面我们介绍单片机硬件电路中有关串口通信的情况。

（1）单片机片内可编程全双工通信电路

1）发送脚：TXD（P3.1）；
2）接收脚：RXD（P3.0）；
3）管脚为 TTL 电平之间通信（0～5V）。
（2）RS232 接口
1）用于与 PC 机或外部通信的接口；
2）通信电平为（－15V～＋15V）；
3）需要用 MAX232 进行电平转换；
4）通信距离 15m 左右，若需要更长距离通信，可采用 RS 485 接口。
单片机中串行通信主要寄存器：
1）数据缓冲寄存器 SBUF；
2）通信方式控制寄存器 SCON：
■ SM0，SM1 串行工作方式控制位。
■ REN：允许串行接收位，1 允许；0 禁止。
■ TI：发送中断标志位，发送完 8 位数据后，由硬件置 1，需用软件清除。
■ RI：接受中断标志位，接收完 8 位数据后，由硬件置 1，需用软件清除。
■ 电源控制寄存器 PCON：
■ SMOD（D7）＝1，串行通信波特率加倍。
■ 定时器工作方式控制器 TMOD。
■ 定时器控制寄存器（TCON）。

看起来还是有点复杂，但是通过实例讲解，很快就会理解了。这里我们只要了解，单片机通过串口进行通信，通信双方需要有相同的波特率，通过发送中断标志和接受中断标志控制发送和接受的进行。

10.1 最简单的单片机串口通信电路

首先，我们来看一个最简单的单片机与单片机通过串口通信的实例，A 单片机向 B 单片机发送数字 1～7，并在 B 单片机显示传送过来的数字 1～7。

下面我们先画出原理图，选择元器件列表见表 10-1。

表 10-1 元器件列表

序 号	元件名称	名称说明	备 注
1	89C52	单片机	51 系列
2	7SEG-COM-AN-BLUE	数码管	共阳，蓝色
3	RES	标准电阻	500Ω
4	MAX232	串口电平转换芯片	

从图 10-1 所示的电路我们看到，两个单片机通过串口引脚连接通信，引脚是交叉连接的，即 A 机的 RXD（P3.0）接 B 机的 TXD（P3.1），A 机的 TXD 接 B 机的 RXD。也可以说，A 单片机是主机，B 单片机是从机。但是，这种连接只适合在电路内部的连接通信。若要连接到外部设备，并且有一定距离，就要采用 RS232 方式或 RS485 方式。

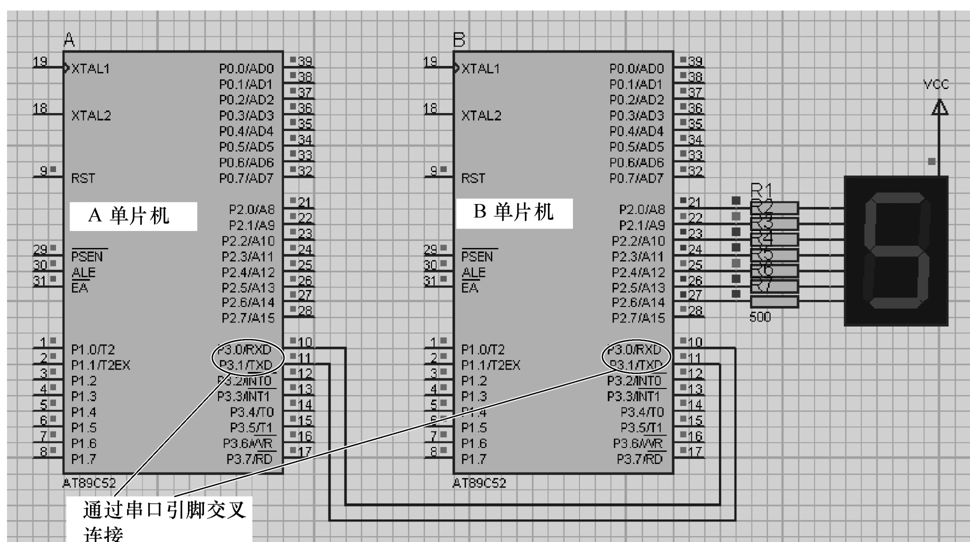


图 10-1　A、B 单片机串口通信

10.2　通过 MAX232 转换的串口通信电路

在更多的情况下，单片机需要连接到外部设备，这就需要通过 MAX232 转换芯片连接。MAX232 芯片的作用就是将 TTL（0 ~ 5V）方式转换为 RS232（ −15V ~ +15V）方式，以便能在较长的距离传输，最长约 15m。

下面我们来看通过 MAX232 转换的串口通信电路，就是在上述电路的基础上，加上 MAX232 芯片，如图 10-2 所示。

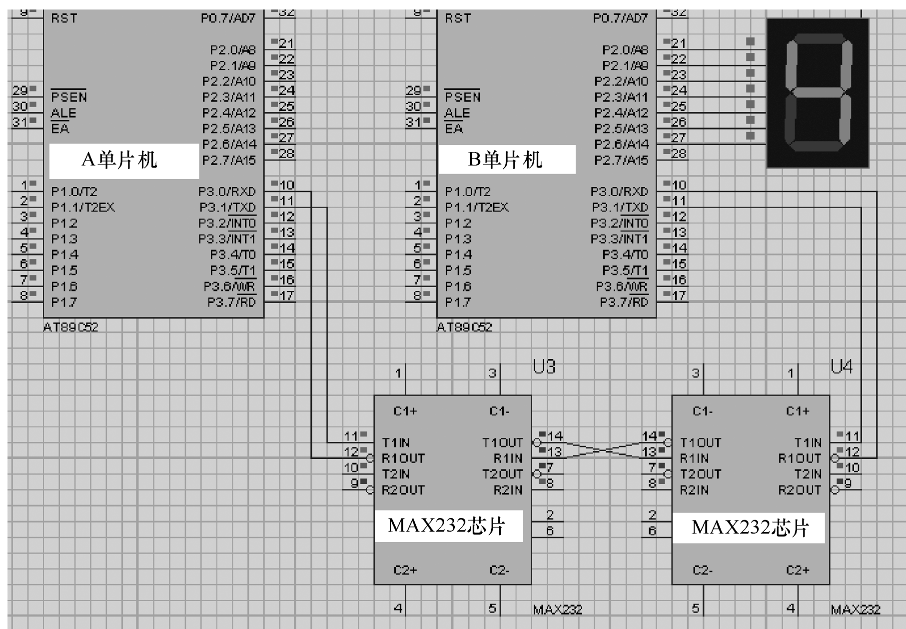


图 10-2　加 MAX232 串口通信

从这个电路看出，单片机串口与 MAX232 的连接都是 T 接 T，R 接 R。即单片机的 TXD 接 MAX232 的 T1IN，RXD 接 R1OUT。而 2 个单片机各自的 MAX232 与对方连接则是交叉的。在 MAX232 之间的连接可以看成是设备外部的连接，距离可达 15m。实际使用中要接一个 9 针插头。

10.3 多机串口通信电路

前面是两个单片机之间的串口通信电路，那么，多个单片机如何处理呢？我们做了一个 3 个单片机串口通信的实例，再多就一样了。就是在上面的基础上再进行复制块操作就可以得到多个单片机之间通信的电路，如图 10-3 和图 10-4 所示。

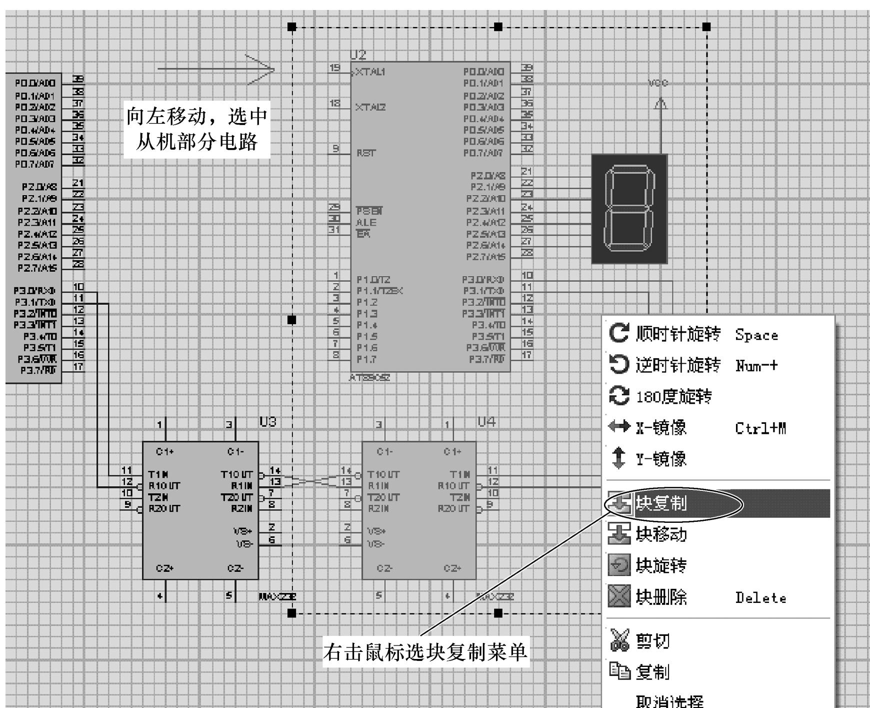


图 10-3 从机部分电路块复制

通过块复制可同时复制多个，非常方便。接着，要适当调整位置，进行连线。将主机与从机的串口通信线断开，水平镜像翻转 MAX232 芯片元件，使位置更紧凑。所有的从机与主机的串口通信线都是交叉连接的，如图 10-5 所示。

这样，就完成了 3 个单片机串口通信的电路。从以上电路可以看出，多个单片机串口通信并不难，只要将串口线与主机交叉连接即可，再多也是一样。

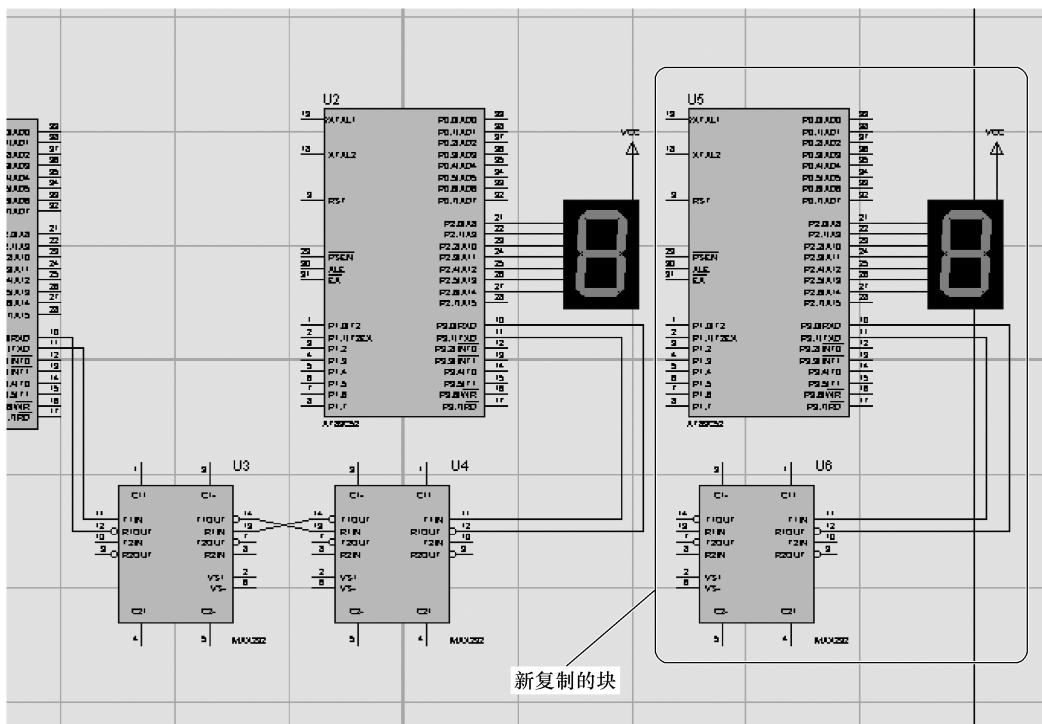


图 10-4　新复制多个块

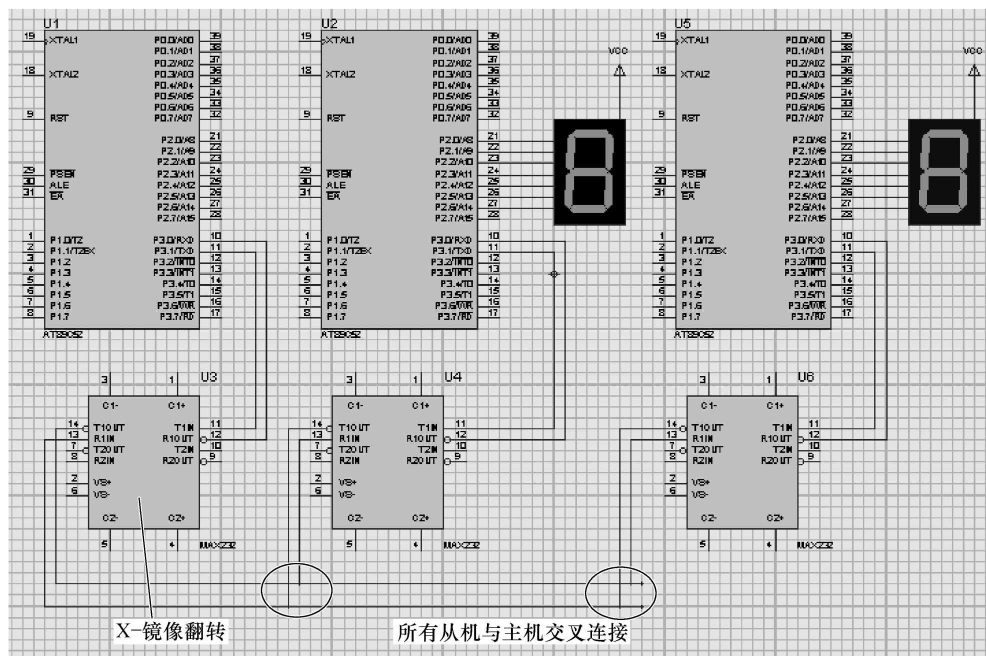


图 10-5　调整位置，串口通信连线

10.4 程序设计与仿真

前面我们进行了两个单片机之间串口通信的电路设计，带有 MAX232 芯片的串口通信和多个单片机之间的串口通信电路设计。虽然电路有所不同，其上运行的程序却可以是一样的，只分主机和从机就行了。下面我们介绍这两个程序，先看主机程序：

```
#include <reg52.h>                    //单片机头文件
void main()
{
    char Buf[] = {1,2,3,4,5,6,7};     //定义一个存放传送内容的数组
    char len = 7;                     //传送数组的长度
    char i = 0;                       //定义循环变量
    SM1 = 1;SM0 = 0;                  //定义串口工作方式 1
    TMOD = 0x20;                      //定义定时器 1 工作方式 2,自动重载,
                                      //8 位计数,用于设置波特率
    TH1 = 0xf3;                       //定时器初值,波特率 2400
    TL1 = 0xf3;
    TI = 0;RI = 0;                    //发送、接受中断标志清零
    TR1 = 1;                          //启动定时器
    SBUF = len;                       //先发送数组长度
    while(TI == 0);                   //等待发送完毕,若发送完毕,则 TI = 1
    TI = 0;                           //发送标志清零
    for(i = 0;i < len;i ++ )          //循环 7 次
    {
        SBUF = Buf[i];                //将发送数据放入缓冲区
        while(TI == 0);               //等待发送完毕
        TI = 0;                       //发送完毕,发送标志清零
    }
    while(1);                         //循环等待
}
```

程序在一开始定义了传送内容的数组和长度，共 7 个数，当然，长度也是 7。然后是串口的工作方式定义，这个计算比较复杂，我们先看 SCON 的定义，见表 10-2。

表 10-2　通信方式控制寄存器 SCON

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

其中的 SM0、SM1 就是决定工作方式的，见表 10-3。

表 10-3　串口工作方式

SM0　SM1	工作方式	功　能	波特率
0　0	方式 0	8 位同步移位	$f_{osc}/12$
0　1	方式 1	10 位 UART	可变
1　0	方式 2	11 位 UART	$f_{osc}/64$ 或 $f_{osc}/32$
1　1	方式 3	11 位 UART	可变

其中，UART 是一种通用串行数据总线，用于异步通信。UART 将接收到的并行数据转换成串行数据来传输。f_{osc}是晶振频率，从波特率这一栏看到，方式 1、3 波特率是可变的，它的计算公式如下：

$$波特率 = \frac{2^{SMOD}}{32 \times} \frac{\times f_{osc}}{12(256 - X)}$$

式中，SMOD 是电源控制寄存器 PCON 的最高位，它控制串行通信波特率加倍，即 SMOD = 1 时，波特率加倍；X 是定时器的初值。

因此，已知波特率、SMOD 和晶振频率f_{osc}，可以求出 X，反之亦然。程序中 X 为 F3 即十进制 243，SMOD = 0，可求得波特率为 2400。

综上所述，程序一开始就是定义串口工作方式和设置串口通信的波特率并启动定时器工作。一般情况下，常用的设置是一样的，只要照搬程序设置部分就行了。

串口通信的动作就是 2 种，发送数据和接收数据。程序写法也几乎是一样的，发送数据：

```
SBUF = Buf[i];                    //将发送数据放入缓冲区
while(TI == 0);                   //等待发送完毕
TI = 0;                           //发送完毕,发送标志清零
```

这个程序，主机只发送数据，没有接收。接收数据也差不多，格式如下：

```
while(RI == 0);                   //等待接收完毕
RI = 0;                           //接收完毕,接收标志清零
Buf[i] = SBUF;                    //将接收数据放入变量中
```

从以上程序可以看出，发送和接收的数据都是放在 SBUF 中的，发送的标志是 TI，当 TI 为 1 时即发送完成，因此，TI 为 0 时要等待，完成后仍然要把 TI 清零，为下一次发送作准备。同样，接收的标志是 RI，当 RI 为 1 时接收完成。

以下是从机的程序：

```
#include <reg52.h>                     //单片机头文件
unsigned char LED_CODES[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                       //共阳数码管码段表
void main()
{
    char Buf[20];                      //存放接收数据数组
    char len;                          //接收数据长度
    char i = 0;                        //循环变量
```

```
    unsigned int k;                         //循环变量
    SM1 =1;SM0 =0;                          //定义串口工作方式 1
    TMOD =0x20;                             //定义定时器 1 工作方式 2,自动重载,
                                            //8 位计数,用于设置波特率
    TH1 =0xf3;                              //定时器初值,波特率 2400
    TL1 =0xf3;
    TI =0;RI =0;                            //发送、接受中断标志清零
    REN =1;                                 //允许串行接收位    1 允许    0 禁止
    TR1 =1;                                 //启动定时器
    while(RI ==0);                          //等待接收完毕
    RI =0;                                  //接收完毕,接收标志清零
    len = SBUF;                             //从缓冲区取接收数据长度
    for(i =0;i < len;i ++ )                 //循环 len 次
    {
        while(RI ==0);
        RI =0;
        Buf[i] =SBUF;                       //接收数据放入数组
    }
    i =0;                                   //循环变量清零
    while(1)                                //无线循环
    {
        P2 =LED_CODES[Buf[i]];              //在数码管显示第 i 个接收数据
        for(k =50000;k!  =0;k--);           //延时
        i ++ ;                              //循环变量加 1
        if(i == len)i =0;                   //循环变量等于数据长度时清零
    }
}
```

我们看到，从机串口通信的定义与主机是一样的，多了一个 REN =1，允许接收数据。接收过程与主机一致，先接收数据长度，然后根据长度循环接收数据。其余就是将接收数据存放在数据，循环显示数据。

主机和从机程序分别新建工程，生成可执行文件，调入单片机即可仿真运行了。

10.5 PC（虚拟终端）与单片机通信

前面我们实现了单片机之间的串口通信。这一节我们介绍一下 PC（Personal Computer，个人计算机）与单片机的串口通信，这里，PC 用虚拟终端替代，虚拟终端是系统提供的虚拟仪器之一。

PC 与单片机串口通信最常用的方法就是用超级终端或串口调试软件实现，在 PC 端也需要设置波特率、数据位、停止位等参数与单片机设置的参数一一对应。首先，我们设计原

理图，选择元器件见表 10-4。

表 10-4　选择元器件

序　　号	元器件名称	名 称 说 明	备　　注
1	89C52	单片机	51 系列
2	7SEG-COM-CA-BLUE	数码管	共阳，蓝色
3	BUTTON	按键	

原理图如图 10-6 所示。

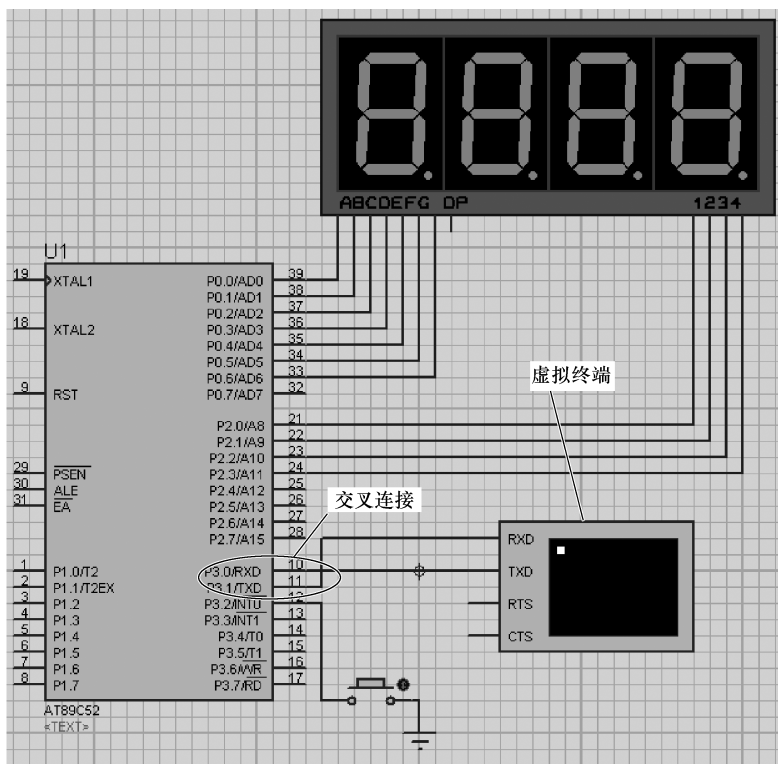


图 10-6　PC 机（虚拟终端）与单片机通信

虚拟终端的选择方法如图 10-7 所示。

首先在模式工具栏选择“虚拟仪器模式”，然后选择“VIRTUAL TERMINAL”即可。虚拟终端的 RXD 与 TXD 与单片机的 RXD 和 TXD 交叉相连。

4 位数码管接在单片机的 P0 口，片选线接在 P2 口的 P2.0 ~ P2.3。按键接在 P3.2 引脚，以便使用外部中断处理。

下面看一下对上述原理图的程序：

```
#include <reg51.h>

unsigned char table1[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                        //共阳数码管码段表
unsigned char table2[] = {0,0,0,0};     //4 位显示数字存放数组
```

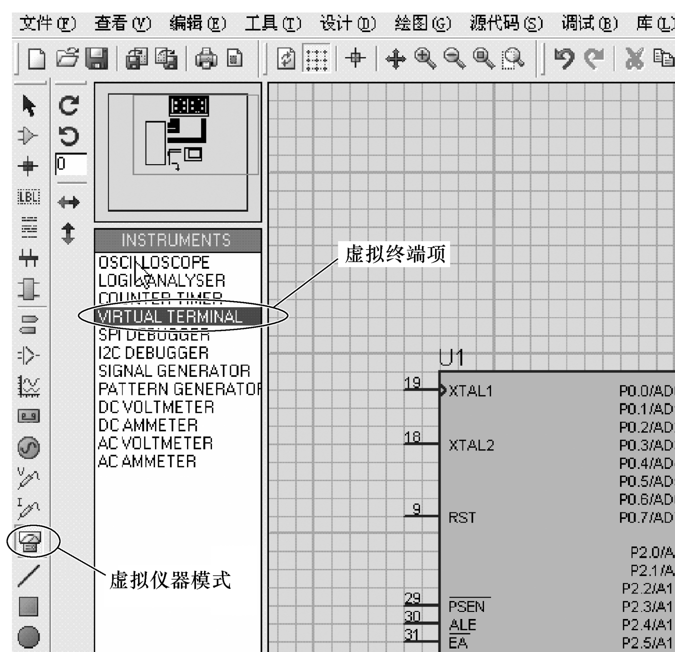


图 10-7 选择虚拟终端

```
unsigned char i;                          //循环变量
unsigned char temp;                       //片选变量

unsigned char dat;                        //接收变量
sbit K = P3^2;                            //定义按键
char code str[ ]  =  "123";               //定义发送数据

void delay(void)                          //延时子程序
{
  int k;
  for(k =0;k <600;k ++ );
}

void display(int k)                       //显示子程序
{
  //table2[3] = k/1000;                   //最高位闲置
  table2[0] = k% 1000/100;                //百位数
  table2[1] = k% 100/10;                  //十位数
  table2[2] = k% 10;                      //个位数
  temp = 0x01;                            //片选初值
  for(i = 0;i < 4;i ++ )                  //循环 4 次
```

```
        {
          P2 = ~temp;                              //送片选值
          P0 = table1[table2[i]];                  //送显示数据位
          temp = temp << 1;                        //片选值左移一位
          delay();                                 //延时
        }
}

void Init_Com(void)                                //串口定义初始化子程序
{
  TMOD = 0x20;                                     //定时器工作方式 2,初值自动装入
  PCON = 0x00;                                     //波特率不倍增
  SCON = 0x50;                                     //串行工作方式 1
  TH1 = 0xFd;                                      //定时器初值高位,波特率 9600
  TL1 = 0xFd;                                      //定时器初值低位
  TR1 = 1;                                         //启动定时器
}

void send_str()                                    //发送字符串子程序
{
      unsigned char i = 0;
      while(str[i] != '\0')                        //直到字符串尾部
      {
            SBUF = str[i];                         //将字符送缓冲区
            while(!TI);                            // 等特数据传送
            TI = 0;                                // 清除数据传送标志
            i++;                                   // 下一个字符
      }
}


void main()
{
  EA = 1;                                          //开总允许中断
  IT0 = 1;EX0 = 1;                                 //开外部中断 0 和外部中断允许
  P0 = 0xff;                                       //P0 口清空
  P2 = 0xff;                                       //P2 口清空
  Init_Com();                                      //调用串口通信初始化子程序
  while(1)
```

```
    {
      if(RI)                                        //如果接收到数据
      {
        dat = SBUF;                                 //将数据从缓冲区读到 dat 中
        RI = 0;                                     //接收标志清零
      }
      display(dat);                                 //数值显示
    }
}

void intersvr0(void)interrupt 0 using 1
{
      send_str();                                   //调用发送字符串子程序
}
```

这个程序的前面部分是定义，包括数码管码段表、变量的定义，和子程序的定义，有延时子程序、显示子程序、串口定义初始化子程序、发送字符串子程序。在程序最后还有一个外部中断处理子程序。

主程序开始进行了外部中断的设置、清空单片机端口和调用串口通信设置初始化子程序，在无限循环中的工作就是接收数据和显示数据。

与前面例子相同，判断是否接收到数据就是判断 RI 中断标志是否为 1，若 RI == 1，就从缓冲区去数据 dat = SBUF，并将 RI 清零，以便下次接收。随后通过调用显示子程序显示接收的数据 display（dat）。

在串口通信初始化中，我们将波特率设置成 9600，因此，在仿真运行前，需要将虚拟终端的属性也进行相应的设置。双击虚拟终端，如图 10-8 所示的方法设置属性。



图 10-8　虚拟终端属性设置

编译程序生成可执行程序，并将程序加入单片机中，运行仿真。要使虚拟终端弹出运行，还必须保证在仿真运行时，在右键单击虚拟终端弹出的菜单中选中“Virtual Terminal”，如图 10-9 所示。

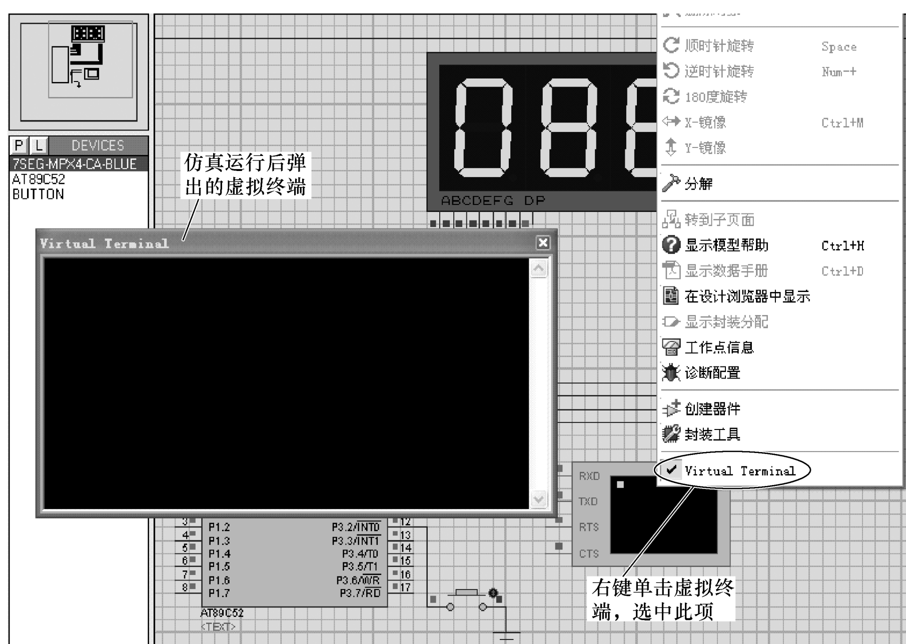


图 10-9　设置虚拟终端运行菜单

在虚拟终端仿真运行中，还可以设置其他参数，以满足各种需要，如图 10-10 所示。



图 10-10　虚拟终端的弹出菜单选项

以下分别对这些选项进行说明：

Clear Screen	虚拟终端清屏
Pause	暂停
Copy	复制
Paste	粘贴
Echo Typed Characters	回显键盘字符
Hex Display Mode	十六进制显示方式
Set Font	设置显示字体字号

首先，我们选择 Echo Typed Characters，这样，当我们敲键盘字符时，就会在虚拟终端上显示出来，并发送给单片机，否则，就不会显示所敲字符，但仍然会发送。下面我们运行仿真，在终端上输入数字 0 ~9，如图 10-11 所示。



图 10-11　在虚拟终端输入字符

我们在键盘上输入的字符与显示的字符是一致，但单片机上接收的数据却不一样，这是因为键盘上的所有字符都属于 ASCII 编码，传输以后还有一个转换的变化造成的。但接收的数据与输入的字符是一一对应的：

输入字符	接收数据
0	112
1	113
2	114
⋮	⋮

可以看出，输入数字与接收数字差 112，修改程序，将接收数据减去 112 就会使接收数据与输入数字一致。修改程序如下：

将　　　　display（dat）;

改为　　　display（dat－112）;

再次运行仿真，如图 10-12 所示。

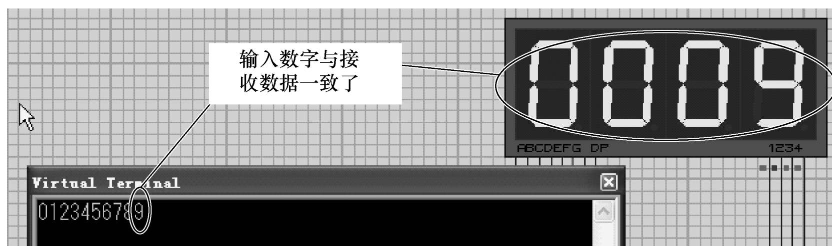


图 10-12 输入数字与接收数据一致

其实，可以通过上述程序，你可以测试出所有键盘字符的值，例如，回车是 13，空格是 96 等等，从而可以用这些值判断用户从键盘上输入什么字符。

我们也可以将输入字符用十六进制显示，将虚拟终端的选项 Hex Display Mode 选中，当我们输入 0 ~9 时就会显示十六进制数字，如图 10 – 13 所示。

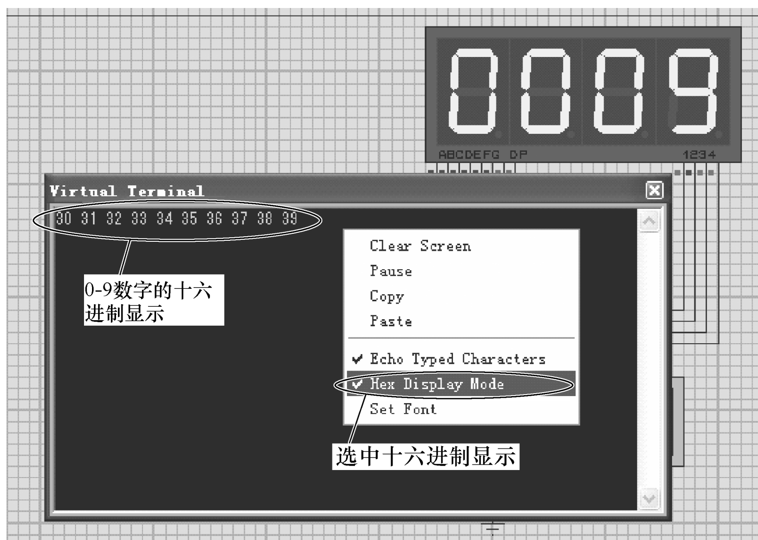


图 10-13 十六进制模式显示字符

现在，我们输入的字符显示的方式不同，在单片机方面接收的数据还是一样的。接着，我们来做一下单片机发送数据给 PC 的操作，通过按钮操作，发送一个字符串。这个字符串是程序中预先定义好的，当按钮按下时，就触发了一个外部中断，中断处理程序就发送了我们定义的字符串“123”，如图 10-14 所示。

这样，我们就实现了单片机与 PC 机通信的仿真。在实际程序设计中，数据转换的值会有所不同，其他都是一致的。

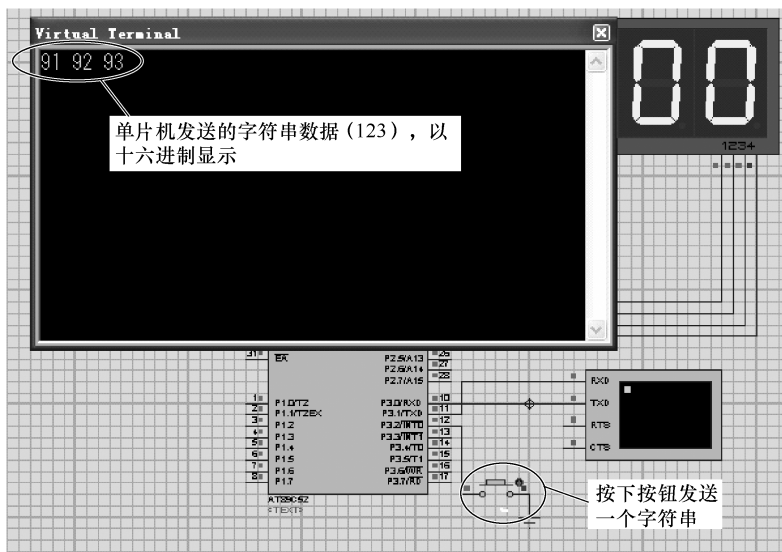


图 10-14　单片机向 PC 发送字符串

第 11 讲

RS485 串口通信

RS485 与 RS232 类似，也是一种串口通信方式。但是，RS485 比 RS232 方式传送的更远、连接的设备更多，它是工业控制领域应用非常普遍的一种通信方式。有了 RS232 串口通信的经验，再来学习 RS485 串口通信就很简单了。

11.1 RS485 串口通信电路

首先我们来设计原理图电路，与 RS232 串口通信一样，先设计一个双机通信的实例，然后再进行多机通信设计，表 11-1 是元器件选择列表。

表 11-1 元器件选择列表

序 号	元器件名称	名 称 说 明	备 注
1	89C52	单片机	51 系列
2	MAX487	电平转换芯片	
3	RES	标准电阻	220Ω
4	LED-BLUE	发光二极管	蓝色

图 11-1 所示为 RS485 双机串口通信电路图，图 11-2 所示为单片机与 MAX487 接线的细节。

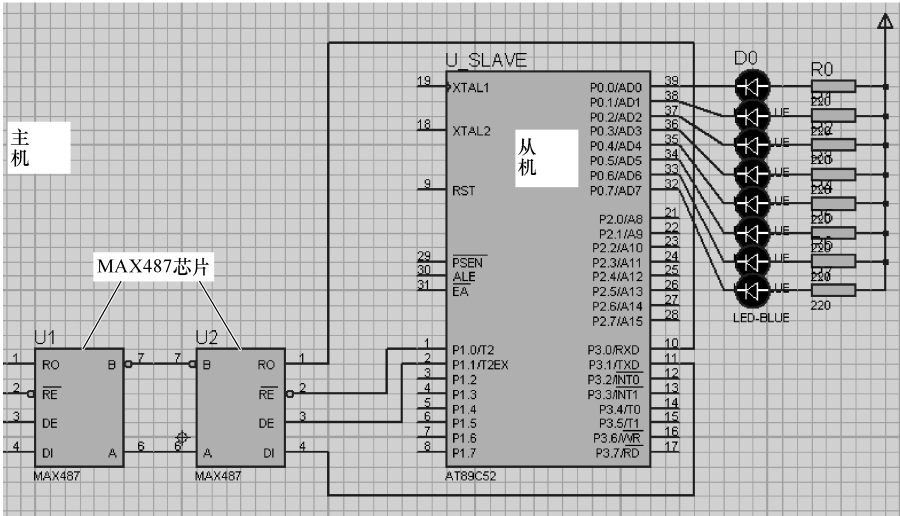


图 11-1 RS485 双机串口通信电路

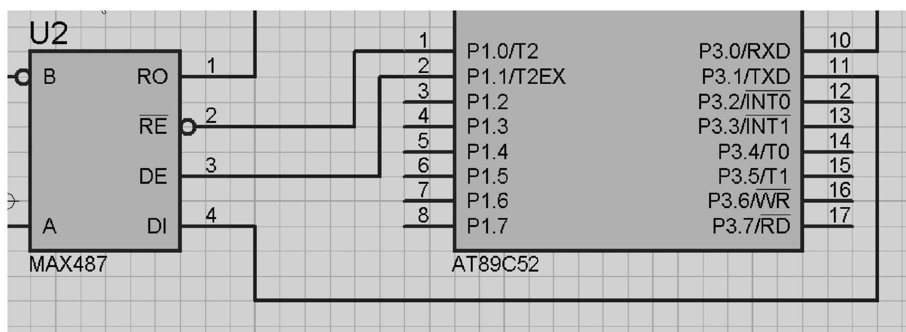


图 11-2　单片机与 MAX487 接线细节

从电路图看出，单片机与 MAX487 的接线，主机与从机是一样的，为了清楚起见，主机部分没有显示。MAX487 的 RO 和 DI 分别接单片机串口通信接口的 RXD 和 TXD，还有两个信号引脚/RE 和 DE 分别接单片机引脚 P1.0 和 P1.1，主机部分接法也是一样的。设备之间通信，MAX487 与 MAX487 连接 A 接 A，B 接 B，没有交叉，这是与 RS232 串口通信不同之处。另外，RS485 通信比 RS232 通信多了 2 个信号，规定接收、发送或同时接收或发送的状态。

为了显示从主机发送过来的数据，在从机 P0 口接了 8 个发光二极管。

11.2　多机 RS485 串口通信电路

与 RS232 串口通信一样，也可以多机通信。在电路制作上，仍可以采用复制的方法，将相同的部分选中，采用块复制的方法，复制多个从机。若读者不清楚怎么做，请参考上一章。图 11-3 所示为多机 RS485 串口通信电路。

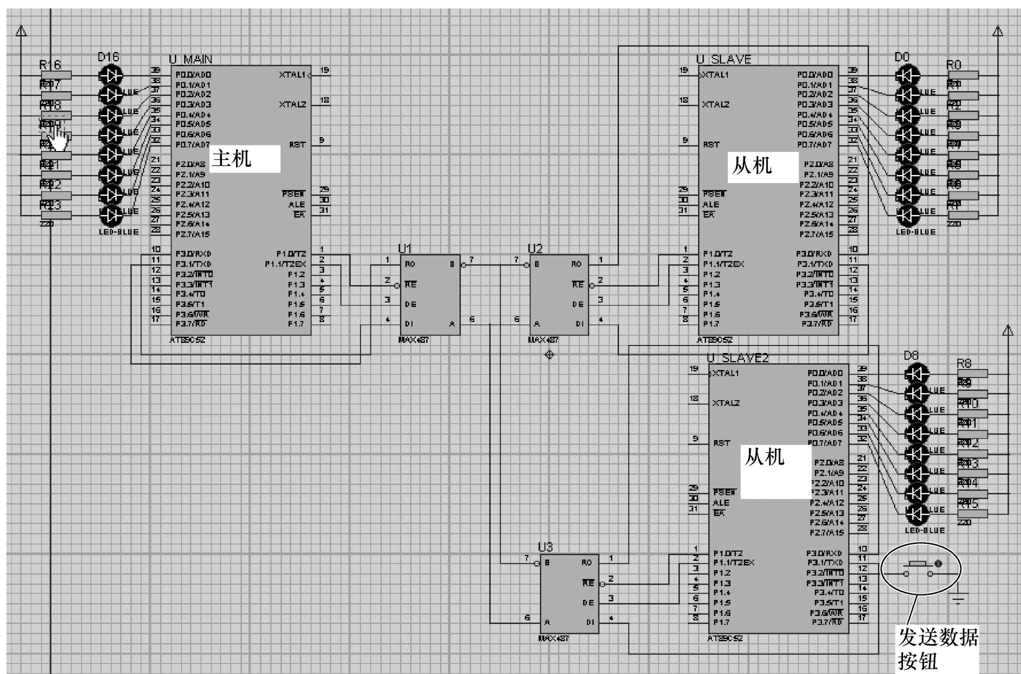


图 11-3　多机 RS485 串口通信电路

从以上电路可以看到，RS485 串口通信的主机和从机完全是对等的，与 RS23 串口通信相似的地方就是都需要通过一个电平转换芯片与其他设备连接。这个连接线可以看成是外线。不同之处就是连接是对等的，不需要交叉。但需要连接两个信号引脚，规定接收还是发送，我们在以下的程序设计中将看到它们的作用。

在电路设计中，每一个单片机都在 P0 口设置了 8 个发光二极管，即每个单片机都可以接收或发送数据。我们在一个从机上设置了一个按钮，以便可以向所有的连接设备发送数据。

11.3　程序设计与仿真

下面我们来看程序设计，首先看主机只发送，从机只接收的程序：

主机程序：

```
#include "reg52.h"

sbit P1_0    =P1^0;                 //接/RE 信号引脚
sbit P1_1    =P1^1;                 //接 DE 信号引脚
void DELAY();                       //声明延时子程序

main()
{
    int tmp;                        //显示变量
    TMOD =0X20;                     //设置 T1 为方式 2
    TH1 =0xFD;                      //设置波特率为 9600
    TL1 =0xFD;
    SCON =0X50;                     //设置串口位方式 1
    PCON =0X00;                     //SMOD =0,频率不倍增
    TR1 =1;                         //定时器 1 开始计数
    P1_0 =1;                        //只发送数据,不接收数据
    P1_1 =1;
    while(1)
    {
        tmp =1;                     //显示变量初值设置
        while(tmp < =128)           //直到最高位为 1
        {
            SBUF = ~tmp;            //求反后送出,发光二极管低电平亮
            while(TI ==0);          //等待发送完成
            TI =0;                  //发送标志清零
            tmp =tmp <<1;           //显示变量左移一位
            DELAY();                //调用延时子程序
```

```
            }
        }
}

void DELAY()                                //延时子程序
{
  int i,j;
  for(i=0;i<100;i++)
      for(j=0;j<1000;j++);
}
```

在程序开始定义了连接/RE、DE 信号的引脚，并在程序中设置/RE、DE 均为 1，即只发送状态。

在主程序中，定义了串口通信的设置，这部分与 RS232 串口通信是一样的，读者可以对照上一章程序。

在无限循环中，就是一个流水灯程序，加上串口发送部分。用变量 tmp 作为显示值，求反后送发送缓冲区，然后左移一位至下一循环，直到最高位为止。因为最高位为 1 即 10000000，就是十进制 128。

发送的步骤与 RS232 串口通信一样，先把数据放入缓冲区，然后等待标志位为 1，最后将标志位清零，以便下一次使用。

再看从机程序：

```
#include "reg52.h"

sbit P1_0    =P1^0;                 //接/RE 信号引脚
sbit P1_1    =P1^1;                 //接 DE 信号引脚
unsigned char k;                    //接收数据变量
void DELAY();

main()
{
    TMOD=0X20;                      //设置 T1 为方式 2
    TH1=0XFD;                       //设置波特率为 9600
    TL1=0XFD;
    SCON=0X50;                      //设置串口位方式 1
    PCON=0X00;
    TR1=1;                          //定时器 1 开始计数
    P1_0=0;                         //设置信号为只读状态
    P1_1=0;
    while(1)
    {
```

```
            while(RI==0);     //等待接收完毕
            k=SBUF;           //读缓冲区数据
            RI=0;             //接收标志位清零
            P0=k;             //数据送 P0 口显示
            DELAY();          //调用延时子程序
    }
}

void DELAY()
{
    int i,j;
    for(i=0;i<100;i++)
      for(j=0;j<1000;j++);
}
```

从机的串口通信设置与主机是一样的。两个信号/RE 和 DE 都设置为 0，即只能接收数据，不能发送数据。

在无限循环中，就是接收数据和显示数据。接收的方法也是与 RS232 串口通信一样，当接收标志位 RI 为 1 时，表示接收完毕，可以从缓冲区取数据，然后将标志位清零，以便下次使用。两个从机的程序是一样的。

运行仿真，我们可以看到，两个从机的 8 个发光二极管循环点亮，即主机发送数据，从机接收数据并显示数据。两个从机接收同样的数据，因此，显示接收的数据也相同。能否有区别地接收呢？比如，一个从机接收 4 位及以下的数据，另一个从机接收 4 位以上的数据。解决这个问题，只要修改程序，在每个从机中加上判断，在接收范围就显示，否则不予显示。现在来修改第 1 个从机程序。

分析问题：若将传过来的数据求反就是 1、2、4、8、16、32、64 、128，4 位及以下的数据就是小于等于 8 的数据。

实现方法：在显示数据前判断数据是否等于 8，若是则显示，否则不与显示。

```
if(~k<=8)            //新加行,将 k 求反比较是否小于等于 8
P0=k;                //原有行,在 P0 口输出数据
```

这样，就实现的显示 4 位及以下的数据。同样，4 位以上也很容易，将判断语句改为：

```
if(~k>8)             //新加行,将 k 求反比较是否大于 8
P0=k;                //原有行,在 P0 口输出数据
```

修改两个从机程序后，再次运行仿真，我们看到，一个从机就只显示 4 位及以下的数据，另一个则显示 4 位以上的数据了。也就是说，所有的数据都同时接收到，但是，只有符合条件的数据才给予显示。

刚才我们做的都是只发送或只接收，能不能既可以发送数据，又能接收数据呢？可以！只要修改两个信号的值就行了，这两个信号的作用定义见表 11-2。

前面，我们已经用了只接收和只发送。现在，要试一下可同时发送和接收的方式。我们需要修改主机程序和第 2 个从机的程序。

表 11-2　MAX487 信号作用

/RE	DE	作　　用
0	0	只接收数据
0	1	接收或发送数据
1	1	只发送数据

主机程序修改如下：

```
#include "reg52.h"

sbit P1_0    =P1^0;
sbit P1_1    =P1^1;
unsigned char k;                //新加行,定义接收变量
void DELAY();

main()
{
    int tmp;
    TMOD =0X20;
    TH1 =0XFD;
    TL1 =0XFD;
    SCON =0X50;
    PCON =0X00;
    TR1 =1;
    P1_0 =0;                    //修改行,将赋值 1 改为 0
    P1_1 =1;
    while(1)
    {
            tmp =1;
            while(tmp < =128)
            {
                SBUF = ~tmp;
                while(TI ==0);
                TI =0;
                tmp = tmp <<1;
                DELAY();
            }
            if(RI)              //新加行,若有接收数据到
            {                   //新加行
              k =SBUF;          //新加行,取缓冲区数据
```

```
            RI = 0;          //新加行,接受标志位清零
            P0 = k;          //新加行,在 P0 口输出数据
            DELAY();         //新加行,调用延时子程序
            P0 = 0xff;       //新加行,P0 口数据清除
        }                    //新加行
    }
}

void DELAY()
{
 int i,j;
 for(i = 0;i < 100;i ++)
    for(j = 0;j < 1000;j ++);
}
```

主机程序修改内容如下：

1）新加一个接收变量 k。

2）将信号 1 1 改为 0 1，使得既能发送，又能接收数据。

3）加上判断项，若有接收数据，就从缓冲区接收，并在 P0 口显示数据，延时一段时间后清除。

再来看可发送数据的从机程序：

```
#include "reg52.h"

sbit P1_0    = P1^0;
sbit P1_1    = P1^1;

sbit S = P3^2;                          //新加行,定义一个按钮
unsigned char k;
void DELAY();

main()
{
    TMOD = 0X20;
    TH1 = 0XFD;
    TL1 = 0XFD;
    SCON = 0X50;
    PCON = 0X00;
    TR1 = 1;
    P1_0 = 0;
    P1_1 = 1;                           //修改行,将 0 改为 1
```

```
    while(1)
    {
            while(RI==0);
            k=SBUF;
            RI=0;
            if(~k>8)
            P0=k;
            DELAY();
            P0=0xff;
           if(S==0)                    //新加行,若按钮按下
           {                           //新加行
                SBUF=~0x81;            //新加行,向缓冲区发送数据
                while(TI==0);          //新加行,等待发送完毕
                TI=0;                  //新加行,发送标志位清零
           }                           //新加行
    }
}

void DELAY()
{
    int i,j;
    for(i=0;i<100;i++)
        for(j=0;j<1000;j++);
}
```

从机程序修改内容如下：

1）定义一个按钮，作为发送数据的操作按钮。

2）将信号0　0改为0　1，使得既可接收数据，又可发送数据。

3）新加判断项，若按钮按下，则发送数据0x81，使得接收方最高和最低位发光二极管点亮编译加载程序，仿真运行，我们发现，按下从机按钮后，主机可以接收数据了，并在8个发光二极管显示出来（最高和最低位点亮）。同时，从机仍在接收主机发送的数据。从机发送的数据，不但主机接收到了，所有的能接收数据的单片机都接收到了，包括发送机自己。

第 12 讲 A-D 转换

A-D 转换（模-数转换）用于实现模拟量到数字量的转换。A-D 转换器件在形式上有两种：串行和并行。并行转换结果可直接获得，串行则需要软件处理后获得，但串行芯片结构简单。本书只介绍串行 A-D 转换芯片的应用。

12.1 A-D 转换原理和电路设计

模数转换主要技术指标有：

■ 分辨率，又称量化间隔，定义为满刻度电压与 2^n 之比，n 为 A-D 转换的位数。

■ 量化误差，有相对误差和绝对误差，绝对误差等于分辨率的一半，即

$$绝对误差 = 分辨率/2$$

■ A-D 转换器类别：计数式、双积分式、逐次逼近式、并行式

这里，我们介绍一个 8 位的串行 A-D 转换芯片 TLC549，其引脚说明如下：

■ 8 位串行 A-D 转换器（若转换最高电压 5V，请问分辨率是多少？量化误差是多少？）

■ REF+　基准电压高端，通常接 V_{CC}

■ REF-　基准电压低端，通常接地

■ AIN　模拟电压输入端

■ /CS　片选端，低电平有效

■ SDO　转换后数据输出端，在时钟信号作用下从高位到低位依次输出

■ SCLK　时钟端

这个芯片共有 6 个引脚，REF+和 REF-分别接电源和接地。AIN 是模拟电压测量输入，当然是接需要测量的模拟电压端。还有 3 个引脚接单片机，SDO 是转换结果输出。$\overline{CS}$是片选信号，SCLK 是时钟信号，2 个信号是单片机控制 A-D 转换芯片进行 A-D 转换的信号。

我们设计一个 A-D 转换的应用，从一个可变电阻上测得变化的电压值，通过发光二极管显示出来。可变电阻一端接电源、另一端接地，滑动变阻端即可变电压输出端接 AIN，即串行模数转换的输入端。元器件选择列表见表 12-1。

图 12-1 所示为电路原理图。

表 12-1　元器件选择列表

序　　号	元器件名称	名 称 说 明	备　　注
1	89C52	单片机	51 系列
2	LED-BLUE	发光二极管	蓝色
3	RESPACK-8	8 位排阻	220Ω
4	TLC549	串行模数转换芯片	8 位
5	POT-HG	可变电阻	1k

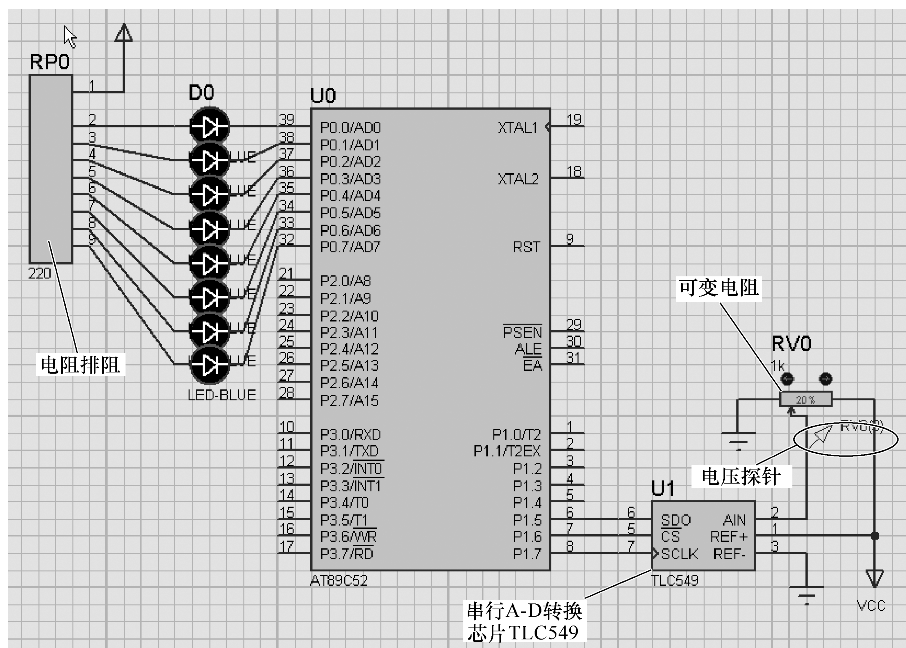


图 12-1　串行模数转换原理图

从电路图我们看到，可变电阻两端分别接电源和接地，滑动变阻端接串行 A-D 转换输入引脚 AIN。在引脚上接了一个电压探针，可实时显示电压变化情况。电压探针从元器件模式工具栏选用，如图 12-2 所示。

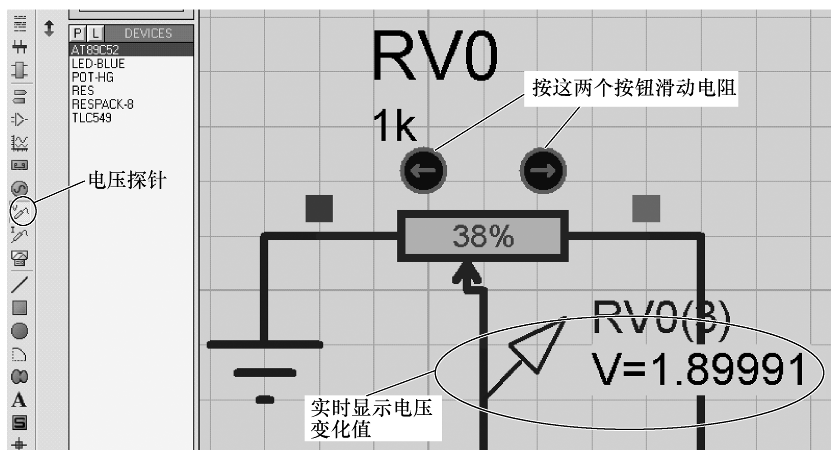


图 12-2　电压探针选用

电压探针的一端接测量点即可，接地端是默认的，无需考虑。

串行模数转换的输出 SDO 接单片机 P1.5 引脚，$\overline{CS}$和 SCLK 分别接 P1.6 和 P1.7 引脚。单片机 P0 口接 8 个发光二极管和一个 8 位的排阻（替代 8 个电阻），排阻公共端接电源。

12.2 程序设计与仿真

根据电路，我们要设计一个程序，从串行 A-D 转换芯片读取电压值，在 P0 口的 8 个发光二极管显示出来。请看以下程序：

```
#include <REG51.h>                    //单片机头文件
#define uchar unsigned char           //定义 uchar 为 unsigned char

//定义 TLC549 串行总线操作端口
sbit    CLK = P1^7;                   //定义 A-D 转换的 SCLK
sbit    DAT = P1^5;                   //定义 A-D 转换的 SDO
sbit    CS = P1^6;                    //定义 A-D 转换的/CS

uchar   bdata   ADCdata;              //定义一个可按位处理的变量
sbit    ADbit = ADCdata^0;            //定义上一个变量的第 1 位


/****************************************************
** 函数名称:    TLC549ADC()
** 函数功能:    读取上一次 A-D 转换的数据,启动下一次 A-D 转换
****************************************************/
uchar   TLC549ADC(void)               //A-D 转换子程序
{
    uchar  i;                         //定义循环变量
    CLK = 0;                          //时钟引脚为 0
    CS = 0;                           //片选引脚为 0
    for(i = 0;i < 8;i ++ )            //循环 8 次
    {
        CLK = 1;                      //时钟引脚为 1
          ADbit = DAT;                //将转换值赋值在存取变量的第 1 位
        ADCdata << = 1;               //存取变量左移一位
        CLK = 0;                      //时钟引脚为 0
    }
        CS = 1;                       //片选引脚为 1
        CLK = 1;                      //时钟引脚为 1
    return(ADCdata);                  //返回 A-D 转换值
```

```
}
void main()                              //主程序
{
    int n;                               //循环变量
    uchar AD_DATA;                       //定义 A-D 转换数据变量
    while(1)
    {
      AD_DATA = TLC549ADC();             //读取当前电压值 A-D 转换数据
        P0 = ~AD_DATA;                   //电压值求反后送 P0 口显示
      for(n = 0;n < 600;n ++);           //延时
    }
}
```

为了简化变量定义，在一开始将 uchar 定义为 unsigned char。下面有一个新的变量类型 bdata，它可以按位处理，如按位赋值、移位等。并将某一位与另一变量联系起来，即 sbit ADbit = ADCdata^0，这样，ADbit 就等于 ADCdata 的第一位，给 ADbit 赋值就是给 ADCdata 的第一位赋值。

然后，分别定义了单片机与串行 A-D 转换芯片连接的 3 个引脚 SDO（DAT）、SCLK（CLK）和 $\overline{CS}$（CS）。

A-D 转换子程序就是将串行模式转换的值读出来，这里就是要把 8 个串行的值变成并行的 8 位数。在读取的时候，要用时钟信号和片选信号控制串行模式转换芯片工作。在一开始时钟和片选信号均为 0，结束后均为 1。在每次循环中时钟信号要从 1 到 0 变化一次。

串行变成并行的方法是：每次把信号读入到变量的第 1 位，然后左移一位，共循环 8 次，就把 8 个串行值变成一个 8 位数了。

主程序中循环体很简单：读取 A-D 转换值、求反后送 P0 口显示、延时。为什么要求反后再显示呢？因为发光二极管 0 发光，1 关闭，它是倒着显示的，求反后正好符合我们的要求。

可以把发光二极管改成数码管，只要将数码管的显示程序加进来就行了。

```
#include <reg52.h>
#define uchar   unsigned char
unsigned char code Tab[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
                                              //数码管码段表
unsigned char Dat[] = {0,0,0,0};              //4 位数字显示存放数组
unsigned char tmp,i;
//定义 TLC549 串行总线操作端口
sbit    CLK = P1^7;
sbit    DAT = P1^5;
sbit    CS = P1^6;

uchar   bdata ADCdata;
```

```
sbit    ADbit = ADCdata^0;

void Delay()
{
  unsigned char i;
    for(i = 0;i < 250;i ++ );
}

/ ****************************************************
** 函数名称:    TLC549ADC()
** 函数功能:    读取上一次 A-D 转换的数据,启动下一次 A-D 转换
**************************************************** /
uchar    TLC549ADC(void)
{
   uchar  i;
   CLK = 0;
   CS = 0;
   for(i = 0;i < 8;i ++ )
   {
       CLK = 1;
         ADbit = DAT;
       ADCdata << = 1;
       CLK = 0;
   }
         CS = 1;
         CLK = 1;
   return(ADCdata);
}
void main()
{
   int n,out;                              //循环变量,输出变量
   float AD_DATA;                          //定义 A-D 转换数据变量
   while(1)
   {
       AD_DATA = TLC549ADC();              //读取当前电压值 A-D 转换数据
       out = AD_DATA * 5 * 1000/256;       //转换成电压值,并扩大 1000 倍
       Dat[0] = out/1000;                  //千位
        Dat[1] = out% 1000/100;            //百位
        Dat[2] = out% 100/10;              //十位
```

```
            Dat[3] = out%10;                          //个位
            tmp = 0x01;                               //数码管片选初值
            for(i = 0;i < 4;i ++ )                    //循环 4 次,显示 4 个数字
            {
                P2 = tmp;                             //设置片选值
                P0 = Tab[Dat[i]];                     //输出数字
                tmp = tmp << 1;                       //片选值左移一位
                Delay();                              //延时
            }
            for(n = 0;n < 600;n ++ );                 //延时
        }
    }
```

这个程序将发个二极管显示改为数码管显示，更加直观。程序中将 8 位的 A-D 转换值在转换成电压值，其公式为 out = AD_DATA * 5 * 1000/256，8 位 A-D 转换的分辨率是 2n 即 256，这里电压最大值是 5V，因此，转换的电压是 AD_ DATA * 5/256，再乘上 1000 是为了在 4 位数码管显示，需要扩大 1000 倍。这样，千位显示的是 0 ~ 5V，其他是小数位。需要注意的是，我们将 AD_ DATA 定义为浮点数 float，是因为运算以后有小数产生，若定义为整数，则小数部分将不能取出而发生错误。数码管的接线采用总线连接的方法，读者若不清楚如何连接总线，可参考第 4 讲内容。图 12-3 所示为仿真运行的结果。

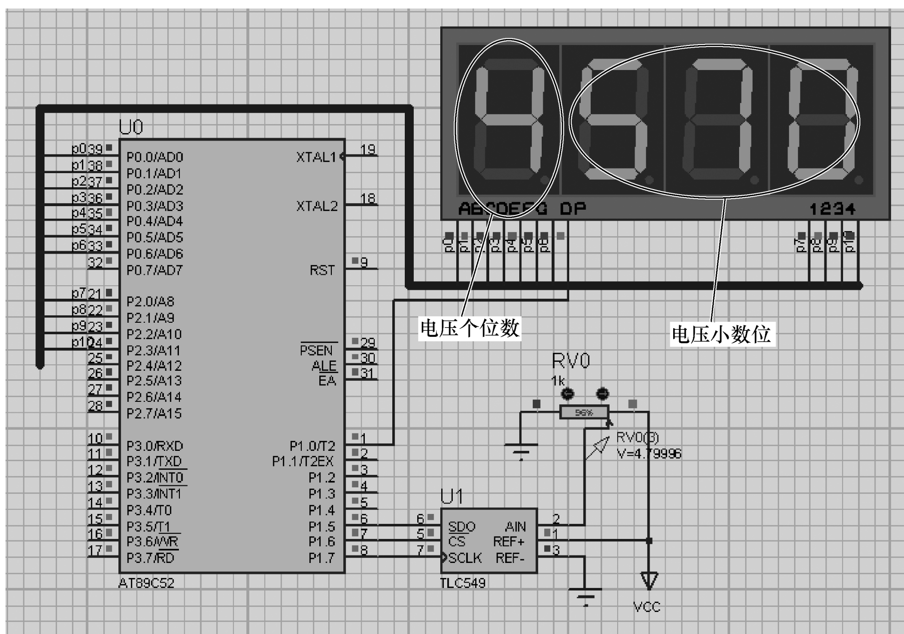


图 12-3　用数码管代替发光二极管显示电压

第
13
讲

并 口 扩 展



导读

单片机只有4个端口P0～P3，为了充分利用这些端口，就需要扩展端口，使得能够接更多的外部设备。这些用于扩展端口的芯片称为可编程I/O接口芯片，单片机最常用的芯片是8255A和8155，这里我们介绍8255A芯片的应用。

13.1 8255A可编程并行I/O接口芯片

我们介绍一下8255A芯片的情况：

1. 8255A结构

1）A口、B口和C口，均为8位I/O数据口。

2）8位数据输出/输入缓冲，锁存器，C口无输入锁存。

3）A、B组控制电路：

■ A组：A口和C口高4位；

■ B组：B口和C口低4位。

4）数据缓冲器：双向三态8位，与单片机数据总线相连。

5）读/写控制逻辑：控制8255A的读写。

2. 8255A外部引脚及功能

1）数据线（8条）：D0～D7，用于传送CPU与8255A之间的数据。

2）控制线和寻址线（6条）：

■ RESET：复位信号，输入高电平有效；

■ $\overline{RD}$，$\overline{WR}$：读/写信号，低电平有效；

■ $\overline{CS}$：片选线，低电平有效；

■ A0，A1：地址输入线，分别用于选择A，B，C口和控制寄存器；

■ I/O口线（24条）：PA0～7，PB0～7，PC0～7；

■ 电源线（2条）：V_{CC}和GND。

3. 8255A 地址表（见表 13-1）

表 13-1　8255A 地址表

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	读操作
0	0	0	1	0	A 口→数据总线
0	1	0	1	0	B 口→数据总线
1	0	0	1	0	C 口→数据总线
					写操作
0	0	1	0	0	数据总线→A 口
0	1	1	0	0	数据总线→B 口
1	0	1	0	0	数据总线→C 口
1	1	1	0	0	数据总线→控制口

4. 8255A 的控制字

1）有 2 个控制字，方式控制和 C 口置/复位控制，以 D7 作为标志：

■ D7 = 1：方式控制；

■ D7 = 0：C 口置/复位控制。

2）控制字定义（P190）

5. 方式控制字

1）D7 = 1；
2）D6、D5 方式选择：
■ 0　0　方式　0
■ 0　1　方式　1　　A 组
■ 1　x　方式　2
3）D4：A 口，1 输入，0 输出；
4）D3：上 C 口，1 输入，0 输出；

5）D2：方式选择；
6）D1：B 口，1 输入，0 输出；　　B 组
7）D0：C 口，1 输入，0 输出；

6. C 口置/复位控制字

1）D7 = 0；

2）D6D5D4：不用；

3）D3D2D1：C 口位选择；

■ 0　0　0　BIT0

■ …

■ 1　1　1　BIT7

4）D0：置位，0　复位　　1　置位

7. 8255A 工作方式

1）方式 0：基本输入/输出方式。

■ A 口，B 口，C 口高 4 位及低 4 位，不需选通；

■ CPU 向 8255A 无条件传送，并在各端口锁存和缓冲。

2）方式 1：选通输入/输出方式。

■ A 口，B 口作为传送 I/O 数据，C 口作为握手联络线。

3）方式 2：双向传送。

■ 仅适用于 A 口；

■ C 口高 5 位作为 A 口联络信号，低 3 位可工作在方式 0，方式 1。

13.2 并口扩展电路设计

前面我们讲了一大堆 8255A 芯片的内部情况，接着，我们来设计并口扩展的原理图。电路元器件选择列表见表 13-2。

表 13-2 元器件选择列表

序　号	元器件名称	名 称 说 明	备　注
1	89C52	单片机	51 系列
2	LED-BLUE	发光二极管	蓝色
3	RESPACK-8	8 位排阻	220Ω
4	8255A	可编程并行 I/O 接口	8 位
5	SWITCH	开关	

图 13-1 所示为电路原理图：

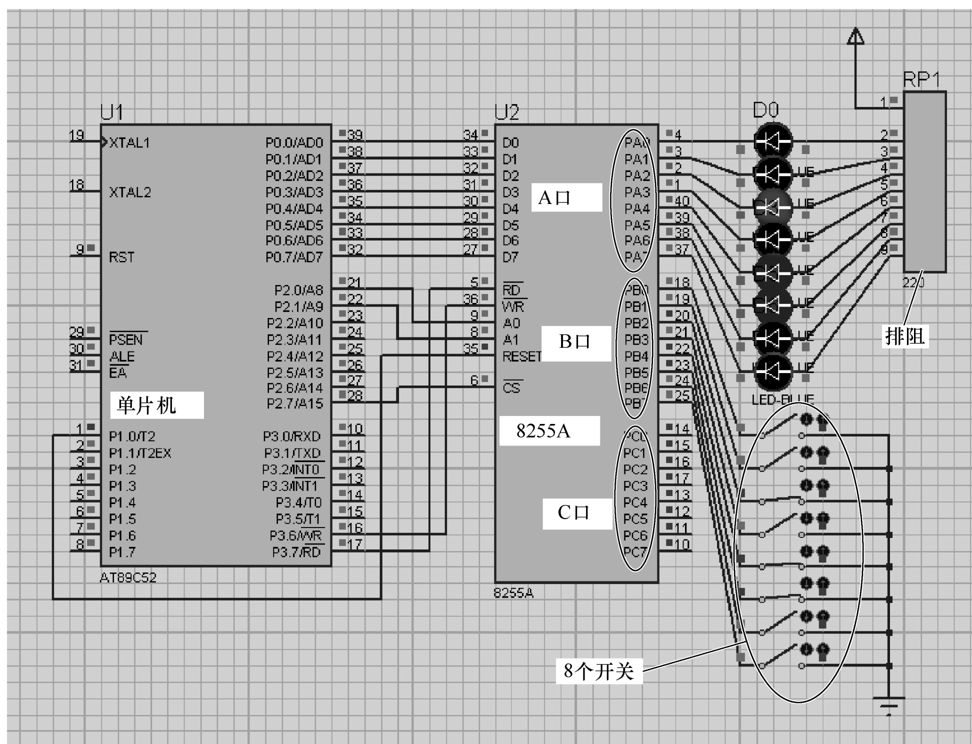


图 13-1 并口扩展原理图

从原理图电路我们看到，单片机 P0 口连接 8255A 数据口，还有 5 根控制线分别连接到单片机的引脚，P2.0、P2.1 连接到 A0、A1，P2.7 连接到片选/CS，P3.6、P3.7 连接到/RD、/WR，P1.0 连接到 RESET。A 口接 8 个发光二极管作为输出，B 口接 8 个开关作为输入，C 口空置。

程序通过判断 B 口开关输入情况，将数据送到 A 口输出，即 B 口开关的每一位对应 A 口的发光二极管，开关开，发光二极管不亮，开关闭合，发光二极管亮。

13.3 程序设计与仿真

根据原理图，我们来设计程序：

```
#include <reg52.h>                    //单片机头文件
#include <absacc.h>                   //扩展地址解析头文件

#define PAC XBYTE[0x7FFF]             //控制地址定义
#define PA  XBYTE[0x7CFF]             //A 口地址定义
#define PB  XBYTE[0x7DFF]             //B 口地址定义
sbit RST = P1^0;                      //复位引脚定义
sbit RD1 = P3^7;                      //读引脚定义
sbit RW = P3^6;                       //写引脚定义

void delay(void)                      //延时子程序
{
    int m,n;
    for(m=0;m<500;m++)
      for(n=0;n<500;n++);
}

void main(void)
{
    unsigned char n;                  //数据存储变量
    RST=1;                            //复位置 1
    RST=0;                            //复位置 0
    PAC=0x82;                         //设控制字
     while(1)
     {
     n=PB;                            //从 B 口读数据
     delay();                         //延时
     PA=n;                            //在 A 口输出数据
     delay();                         //延时
```

```
    }
}
```

这个程序的头文件多了一个 absacc. h，用于解析地址。为了控制使用8255A 芯片，需要定义控制端口、A 口、B 口和 C 口的地址。地址的设置与硬件接线有关，取决于 8255A 的 A0、A1 接在单片机的哪个引脚。

这里需要强调一下，单片机的 P0、P2 口合起来可以组成一个 16 位的地址，由 XBYTE [ ] 定义，P2 口为高 8 位，P0 口为低 8 位。A1A0 地址的已定义在表 13-1 中，当 A1A0 为 00 时就是 A 口，01 时就是 B 口，11 时就是控制口，片选 $\overline{CS}$ 始终为 0。设其他未定义引脚都为 1，接线地址见表 13-3。

表 13-3 8255A 地址设置表

单片机引脚	P2. 7	P2. 6	P2. 5	P2. 4	P2. 3	P2. 2	P2. 1	P2. 0	P0. 7	P0. 6	P0. 5	P0. 4	P0. 3	P0. 2	P0. 1	P1. 7
8255A 引脚	$\overline{CS}$						A1	A0								
A 口	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1
B 口	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
控制口	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
十六进制值	7				X				F				F			

根据这个表，我们很容易看出每个地址的设置值：

A 口　　　7CFF

B 口　　　7DFF

控制口　　7FFF

从控制字定义可知，D6、D5 决定控制方式，D4 决定 A 口，D1 决定 B 口，见表 13-4。

表 13-4 控制字设置

数据位	D7	D6	D5	D4	D3	D2	D1	D0
数值	1	0	0	0	0	0	1	0
十六进制	8				2			
功能	方式控制	方式 0		A 口输出			B 口输入	

在循环程序中，不断地从 B 口读入数据到变量 n 中，延时一段时间，然后将 n 输出到 A 口，再延时一段时间，周而复始。因此，当 B 口的开关量变化时，就会反映到 A 口的输出来。

第14讲

步进电动机控制



导读

本讲讨论了如何用单片机控制步进电动机的旋转，控制电路、步进电动机驱动芯片和步进电动机的连接，以及步进电动机旋转控制的编程方法。

14.1 步进电动机控制原理

步进电动机是一种异步电动机，它的工作原理是利用电子电路，将直流电变成分时供电的，多相时序控制电流，用这种电流为步进电动机供电，步进电动机才能正常工作，驱动器就是为步进电动机分时供电的，多相时序控制器。虽然步进电动机已被广泛地应用，但步进电动机并不能像普通的直流电动机，交流电动机在常规下使用。它必须由双环形脉冲信号、功率驱动电路等组成控制系统方可使用。因此用好步进电动机却非易事，它涉及机械、电动机、电子及计算机等许多专业知识。

步进电动机作为执行元件，是机电一体化的关键产品之一，广泛应用在各种自动化控制系统中。随着微电子和计算机技术的发展，步进电动机的需求量与日俱增，在各个国民经济领域都有应用。

现在比较常用的步进电动机包括反应式步进电动机（VR）、永磁式步进电动机（PM）、混合式步进电动机（HB）和单相式步进电动机等。

永磁式步进电动机一般为两相，转矩和体积较小，步进角一般为7.5°或15°。永磁式步进电动机输出力矩大，动态性能好，但步距角大。

反应式步进电动机一般为三相，可实现大转矩输出，步进角一般为1.5°，但噪声和振动都很大。反应式步进电动机的转子磁路由软磁材料制成，定子上有多相励磁绕组，利用磁导的变化产生转矩。反应式步进电动机结构简单，生产成本低，步距角小；但动态性能差。

混合式步进电动机综合了反应式、永磁式步进电动机两者的优点，它的步距角小，出力大，动态性能好，是目前性能最高的步进电动机。它有时也称作永磁感应子式步进电动机。它又分为两相和五相：两相步进角一般为1.8°，而五相步进角一般为0.72°。这种步进电动机的应用最为广泛。

由于反应式步进电动机工作原理比较简单。下面先叙述三相反应式步进电动机原理。

1. 三相反应式步进电动机原理

（1）结构

电动机转子均匀分布着很多小齿，定子齿有三个励磁绕组，其几何轴线依次分别与转子齿轴线错开。0、1/3て、2/3て（相邻两转子齿轴线间的距离为齿距以て表示），即 A 与齿 1 相对齐，B 与齿 2 向右错开 1/3て，C 与齿 3 向右错开 2/3て，A′与齿 5 相对齐，（A′就是 A，齿 5 就是齿 1）。

（2）旋转

如 A 相通电，B，C 相不通电时，由于磁场作用，齿 1 与 A 对齐，（转子不受任何力，以下均同）。如 B 相通电，A，C 相不通电时，齿 2 应与 B 对齐，此时转子向右移过 1/3て，此时齿 3 与 C 偏移为 1/3て，齿 4 与 A 偏移（て－1/3て）＝2/3て。如 C 相通电，A，B 相不通电，齿 3 应与 C 对齐，此时转子又向右移过 1/3て，此时齿 4 与 A 偏移为 1/3て对齐。如 A 相通电，B，C 相不通电，齿 4 与 A 对齐，转子又向右移过 1/3て 这样经过 A、B、C、A 分别通电状态，齿 4（即齿 1 前一齿）移到 A 相，电动机转子向右转过一个齿距，如果不断地按 A，B，C，A……通电，电动机就每步（每脉冲）1/3て，向右旋转。如按 A，C，B，A……通电，电动机就反转。由此可见：电动机的位置和速度由导电次数（脉冲数）和频率成一一对应关系。而方向由导电顺序决定。不过，出于对力矩、平稳、噪声及减少角度等方面考虑。往往采用 A—AB—B—BC—C—CA—A 这种导电状态，这样将原来每步 1/3て改变为1/6て。甚至于通过二相电流不同的组合，使其 1/3て变为 1/12て，1/24て，这就是电动机细分驱动的基本理论依据。不难推出：电动机定子上有 m 相励磁绕阻，其轴线分别与转子齿轴线偏移 1/m，2/m…（m－1）/m，1。并且导电按一定的相序电动机就能正反转被控制——这是旋转的物理条件。只要符合这一条件我们理论上可以制造任何相的步进电动机，出于成本等多方面考虑，市场上一般以二、三、四、五相为多。

2. 感应子式步进电动机原理：

（1）特点

感应子式与传统的反应式相比，结构上转子加有永磁体，以提供软磁材料的工作点，而定子励磁只需提供变化的磁场而不必提供磁材料工作点的耗能，因此该电机效率高，电流小，发热低。因永磁体的存在，该电机具有较强的反电动势，其自身阻尼作用比较好，使其在运转过程中比较平稳、噪声低、低频振动小。感应子式某种程度上可以看作是低速同步的电动机。一个四相电动机可以做四相运行，也可以做二相运行（必须采用双极电压驱动），而反应式电动机则不能如此。例如，四相，八相运行（A—AB—B—BC—C—CD—D—DA—A）完全可以采用二相八拍运行方式，不难发现其条件为 $C=\overline{A}$，$D=\overline{B}$，一个二相电机的内部绕组与四相电动机完全一致，小功率电动机一般直接接为二相，而功率大一点的电动机，为了方便使用，灵活改变电动机的动态特点，往往将其外部接线为八根引线（四相），这样使用时，既可以作四相电动机使用，可以作二相电动机绕组串联或并联使用。

（2）分类

感应子式电机以相数可分为：二相电动机、三相电动机、四相电动机、五相电动机等。以机座号（电动机外径）可分为：42BYG（BYG 为感应子式步进电动机代号）、57BYG、86BYG、110BYG、（国际标准），而像 70BYG、90BYG、130BYG 等均为国内标准。

（3）步进电动机的静态指标术语

1）相数：产生不同对极 N、S 磁场的励磁线圈对数，常用 m 表示。

2）拍数：完成一个磁场周期性变化所需脉冲数或导电状态用 n 表示，或指电动机转过一个齿距角所需脉冲数，以四相电机为例，有四相四拍运行方式即 AB—BC—CD—DA—AB，四相八拍运行方式即 A—AB—B—BC—C—CD—D—DA—A。

3）步距角：对应一个脉冲信号，电动机转子转过的角位移用 θ 表示。$\theta = 360°/$（转子齿数 * 运行拍数），以常规二、四相，转子齿为 50 齿电动机为例。四拍运行时步距角为 $\theta = 360°/(50 \times 4) = 1.8°$（俗称整步），八拍运行时步距角为 $\theta = 360°/(50 \times 8) = 0.9°$（俗称半步）。

4）定位转矩：电动机在不通电状态下，电动机转子自身的锁定力矩（由磁场齿形的谐波以及机械误差造成的）。

5）静转矩：电动机在额定静态电作用下，电动机不作旋转运动时，电动机转轴的锁定转矩。此转矩是衡量电动机体积（几何尺寸）的标准，与驱动电压及驱动电源等无关。虽然静转矩与电磁励磁安匝数成正比，与定齿转子间的气隙有关，但过分采用减小气隙，增加励磁安匝来提高静转矩是不可取的，这样会造成电动机的发热及机械噪声。

（4）动态指标及术语

1）步距角精度：步进电动机每转过一个步距角的实际值与理论值的误差。用百分比表示：误差/步距角 * 100%。不同运行拍数其值不同，四拍运行时应在 5% 之内，八拍运行时应在 15% 以内。

2）失步：电动机运转时运转的步数，不等于理论上的步数。称之为失步。

3）失调角：转子齿轴线偏移定子齿轴线的角度，电动机运转必存在失调角，由失调角产生的误差，采用细分驱动是不能解决的。

4）最大空载起动频率：电动机在某种驱动形式、电压及额定电流下，在不加负载的情况下，能够直接起动的最大频率。

5）最大空载的运行频率：电动机在某种驱动形式，电压及额定电流下，电动机不带负载的最高转速频率。

6）运行矩频特性：电动机在某种测试条件下测得运行中输出力矩与频率关系的曲线称为运行矩频特性，这是电动机诸多动态曲线中最重要的，也是电动机选择的根本依据。其他特性还有惯频特性、起动频率特性等。电动机一旦选定，电动机的静转矩确定，而动态转矩却不然，电动机的动态力矩取决于电动机运行时的平均电流（而非静态电流），平均电流越大，电动机输出转矩越大，即电动机的频率特性越硬。

7）电动机的共振点：步进电动机均有固定的共振区域，二、四相感应子式的共振区一般在 180 ~ 250Hz 之间（步距角 1.8°）或在 400Hz 左右（步距角为 0.9°），电动机驱动电压越高，电动机电流越大，负载越轻，电机体积越小，则共振区向上偏移，反之亦然，为使电机输出转矩大，不失步和整个系统的噪音降低，一般工作点均应偏移共振区较多。

8）电动机正反转控制：当电动机绕组通电时序为 AB—BC—CD—DA 或（）时为正转，通电时序为 DA—CD—BC—AB 或（$A\overline{B}$—$\overline{A}\,\overline{B}$—$\overline{A}B$—AB）时为反转。

14.2　步进电动机控制电路

首先我们来设计步进电动机的驱动电路，选择元器件见表 14-1。

表 14-1　元器件列表

序　　号	元器件名称	名 称 说 明	备　　注
1	89C52	单片机	51 系列
2	MOTOR-STEPPER	步进电动机	四相
3	RES	电阻	20Ω
4	ULN2003A	步进电动机驱动芯片	
5	SWITCH	开关	

设计电路图如图 14-1 所示。

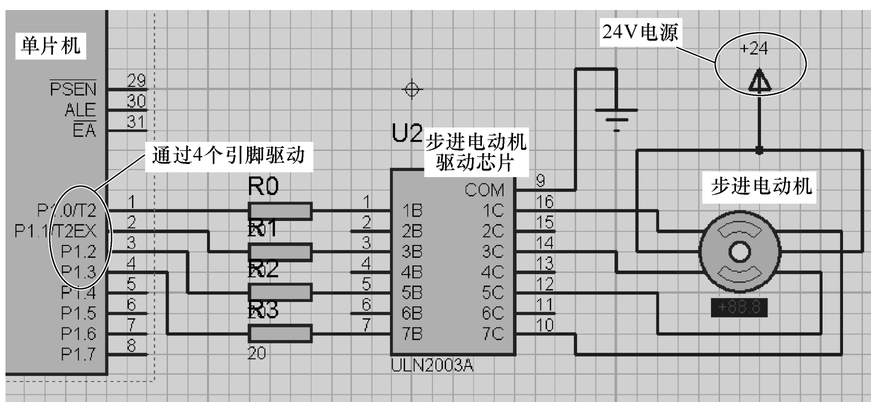


图 14-1　单片机驱动步进电动机电路

电路看起来还是比较简单的，单片机的 4 个引脚，P1.0 ~ P1.3，通过 20Ω 电阻连接到驱动芯片 ULN2003A，驱动芯片的 4 个引脚接步进电机线圈相引脚（见图 14-2），公共端 COM 接地。步进电动机的电源端接 24V 电源（电源端子需调整至 24V）。

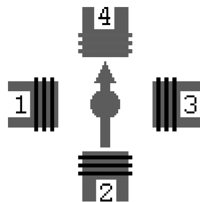


图 14-2　步进电动机内部结构

14.3　程序设计与仿真

单片机驱动步进电动机的程序设计如下：

```
#include "reg51.h"                          //单片机头文件
void DELAY();                               //声明延时子程序
main()
{
```

```
    while(1)
    {
        P1 = 0x08;          //A          //A 相高电平
        DELAY();
        P1 = 0x0C;          //AB         //AB 相高电平
        DELAY();
        P1 = 0x04;          //B          //B 相高电平
        DELAY();
        P1 = 0x06;          //BC         //BC 相高电平
        DELAY();
        P1 = 0x02;          //C          //C 相高电平
        DELAY();
        P1 = 0x03;          //CD         //CD 相高电平
        DELAY();
        P1 = 0x01;          //D          //D 相高电平
        DELAY();
        P1 = 0x09;          //DA         //DA 相高电平
        DELAY();
      }
}
void DELAY()                             //延时子程序
{
    int i,j;
      for(i = 0;i < 240;i ++ )
          for(j = 0;j < 200;j ++ );
}
```

程序通过变换步进电动机的通电相，A—AB—B—BC—C—CD—D—DA—A，按照四相八拍运行方式周而复始使的步进电动机转动起来。我们将上述通电相情况画成表 14-2，效果如图 14-3 所示。

表 14-2　单片机引脚数值与步进电动机通电相位

数值＼位	P1.3	P1.2	P1.1	P1.0	通电相
P1 = 0x08	1	0	0	0	A
P1 = 0x0C	1	1	0	0	AB
P1 = 0x04	0	1	0	0	B
P1 = 0x06	0	1	1	0	BC
P1 = 0x02	0	0	1	0	C
P1 = 0x03	0	0	1	1	CD
P1 = 0x01	0	0	0	1	D
P1 = 0x09	1	0	0	1	DA

从表格可以看出，高电平 1 的走向决定的步进电动机的旋转方向，调整 P1 端口的输出值，就可以使步进电动机按照相反的方向转动。因此，可以在电路中设置一个开关，根据开关的位置，决定让步进电动机旋转方向。

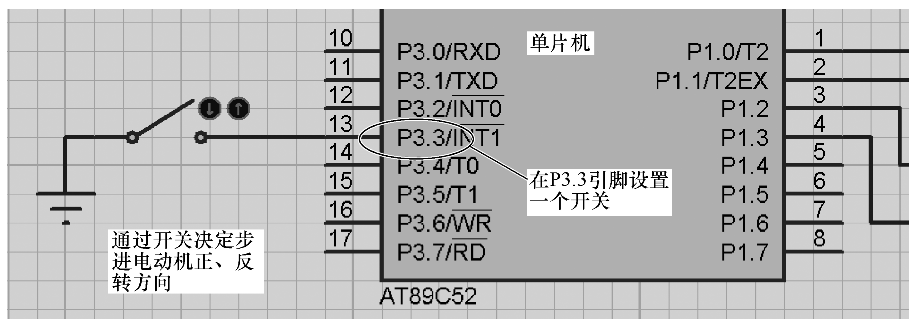


图 14-3　单片机控制步进电动机加开关

程序如下：

```
#include "reg51.h"
sbit S = P3^3;
void DELAY();

main()
{
    while(1)
    {
        if(S)                    //如果开关打开,步进电动机正向转动
        {
        P1 =0x08;                //A
        DELAY();
        P1 =0x0C;                //AB
        DELAY();
        P1 =0x04;                //B
        DELAY();
        P1 =0x06;                //BC
        DELAY();
        P1 =0x02;                //C
        DELAY();
        P1 =0x03;                //CD
        DELAY();
        P1 =0x01;                //D
        DELAY();
        P1 =0x09;                //DA
```

```
        DELAY( );
        }
      Else                          //如果开关闭合,步进电动机反向转动
      {
        P1 = 0x09;                  //DA
        DELAY( );
        P1 = 0x01;                  //D
        DELAY( );
        P1 = 0x03;                  //CD
        DELAY( );
        P1 = 0x02;                  //C
        DELAY( );
        P1 = 0x06;                  //BC
        DELAY( );
        P1 = 0x04;                  //B
        DELAY( );
        P1 = 0x0c;                  //AB
        DELAY( );
        P1 = 0x08;                  //A
        DELAY( );
        }

    }
}
void DELAY( )
{
   int i,j;
     for( i = 0;i < 240;i ++ )
     for( j = 0;j < 200;j ++ );
}
```

读者可以根据程序列出步进电动机反向走向的相位列表。

前面我们只做到了步进电动机的正反转旋转，下面我们要实现用按钮实现步进，即按一下，步进电机前进或后退一步。只要在 P3. 2、P3. 3 分别连接一个按钮，并用外部中断方式处理步进电动机的旋转，程序如下：

```
#include "reg51. h"                    //单片机头文件
void DELAY( );                         //声明延时子程序
sbit S1 = P3^2;                        //设置按钮 S1,触发外部中断 0
sbit S2 = P3^3;                        //设置按钮 S2,触发外部中断 1
int x = 0x01;
```

```
main()
{
    EA = 1;                         //开总允许开关
    IT0 = 1;EX0 = 1;                //开外部中断 0 和外部中断 0 允许分开关
    IT1 = 1;EX1 = 1;
    while(1);                       //无限循环等待
}
void DELAY()                        //延时子程序
{
    int i,j;
     for(i = 0;i < 240;i ++ )
       for(j = 0;j < 200;j ++ );
}

void intersvr0(void)interrupt 0 using 1//外部中断 0 处理子程序
{
    if(x == 1)
     x = 3;
    else if(x == 3)
     x = 2;
    else if(x == 2)
     x = 6;
    else if(x == 6)
     x = 4;
    else if(x == 4)
     x = 12;
    else if(x == 12)
     x = 8;
    else if(x == 8)
     x = 9;
    else if(x == 9)
     x = 1;
    P1 = x;


}

void intersvr1(void)interrupt 2 using 1//外部中断 1 处理子程序
{
    if(x == 8)
```

```
    x = 12;
   else if(x == 12)
    x = 4;
   else if(x == 4)
    x = 6;
   else if(x == 6)
    x = 2;
   else if(x == 2)
    x = 3;
   else if(x == 3)
    x = 1;
   else if(x == 1)
    x = 9;
   else if(x == 9)
    x = 8;
   P1 = x;
}
```

这个程序在 P3.2 和 P3.3 引脚定义了两个按钮，并设置了外部中断 0 和外部中断 1。当按下 S1 按钮，就触发外部中断 0，有外部中断程序处理。外部中断程序首先判断步进电机导通相位，然后给出下一个导通相位值，使步进电动机按逆时针方向转动一步。同理，按下 S2 按钮，则向顺时针方向转动一步。

由于是四相八拍，走一步就是 1/8 圈，所以走 8 步就是 1 圈，非常准确。完整的电路及仿真如图 14-4 所示。

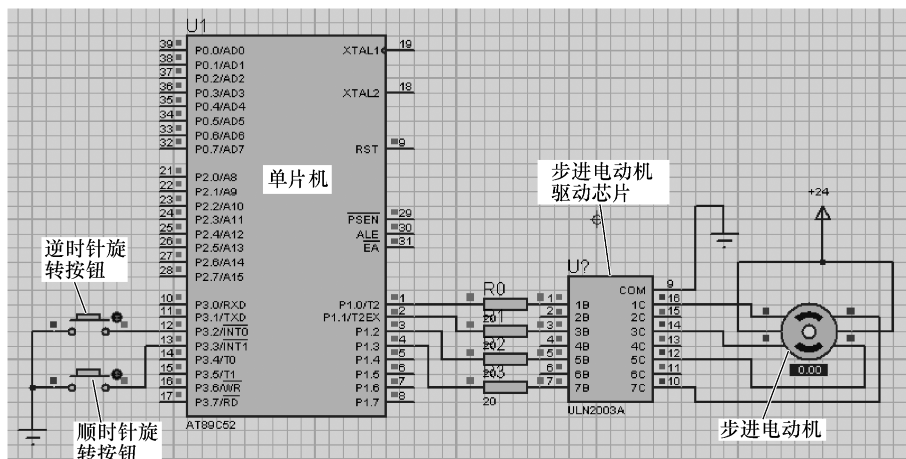


图 14-4　步进电动机单步控制

第15讲 直流电动机控制

 导读

本讲讨论单片机对直流电动机的控制。首先讲直流电动机的控制原理，然后讨论单片机实现占空比输出，进而实现对直流电动机的控制。

15.1 直流电动机控制原理

电动机分为交流电动机和直流电动机两大类。长期以来，直流电动机以其良好的线性特性、优异的控制性能、较强的过载能力成为大多数变速运动控制和闭环位置伺服控制系统的最佳选择，一直处在调速领域主导地位。传统的直流电动机调速方法很多，如调压调速、弱磁调速等，它们存在着调速响应慢、精度差、调速装置复杂等缺点。随着全控式电力电子器件技术的发展，以大功率晶体管作为开关器件的直流脉宽调制（Pulse Width Modulation，PWM）调速系统已成为直流调速系统的主要发展方向。

在 PWM 调速系统中，一般可以采用定宽调频、调宽调频、定频调宽三种方法改变控制脉冲的占空比，但是前两种方法在调速时改变了控制脉宽的周期，从而引起控制脉冲频率的改变，当该频率与系统的固有频率接近时将会引起振荡。为避免之，设计采用定频调宽改变占空比的方法来调节直流电动机电枢两端电压。

定频调宽法的基本原理是按一个固定频率来接通和断开电源，并根据需要改变一个周期内接通和断开的时间比（占空比）来改变直流电动机电枢上电压的“占空比”，从而改变平均电压，控制电动机的转速。在 PWM 调速系统中，当电动机通电时其速度增加，电动机断电时其速度减低。只要按照一定的规律改变通、断电的时间，即可控制电动机转速。而且采用 PWM 技术构成的无级调速系统，起停时对直流系统无冲击，并且具有起动功耗小、运行稳定的优点。为了说明问题，现假定电动机始终接通电源时，电动机最大转速为 V_{max}，占空比为 $D=t/T$，则电动机的平均速度 $V_d=DV_{max}$，由公式可知，当改变占空比 $D=t/T$ 时，就可以得到不同的电动机平均速度 V_d，从而达到调速的目的。在一般应用中，可将平均速度与占空比 D 近似地看成线性关系。

15.2 用定时器控制占空比

从直流电动机的控制要求出发，我们需要控制高电平输出的个数，即在某个周期内输出

高电平的比例，称为 PWM 占空比。PWM 占空比越大，电动机获得的能量就越大，电动机就转动的越快。

用单片机控制占空比的常用方法是用定时器来实现，用定时器输出固定的脉冲宽度，以此来计数，确定在某个周期内输出的脉冲个数。

占空比控制程序如下：

```
#include"reg52.h"                                    //单片机头文件

sbit P1_0 = P1^0;                                    //脉冲输出引脚
sbit P1_1 = P1^1;                                    //按钮 1 引脚
sbit P1_2 = P1^2;                                    //按钮 2 引脚

unsigned char PWMH;                                  //高电平脉冲的个数
unsigned char PWM;                                   //PWM 周期
unsigned char COUNTER;                               //计数变量

void K1CHECK();                                      //按钮 1 处理程序
void K2CHECK();                                      //按钮 2 处理程序

void INTTO()interrupt 1                              //定时中断 0 处理程序
{
  COUNTER++;                                         //计数值加 1
  if((COUNTER!=PWMH)&&(COUNTER==PWM))                //如果计数值不等于设定的高
  {                                                  //电平脉冲数,但计数值已到
                                                     //达周期数
      COUNTER=1;                                     //计数器复位
      P1_0=1;                                        //P1.0 为高电平
  }
  else if(COUNTER==PWMH)                             //如果计数值等于设定的高电
                                                     //平脉冲数
      P1_0=0;                                        //P1.0 变为低电平
}

main()
{
  PWMH=0x02;                                         //设置高电平脉冲数
  COUNTER=0x01;                                      //计数初值
  PWM=0x15;                                          //设置计数周期数
  TMOD=0x02;                                         //定时器 0 在模式 2 下工作
  TL0=0x38;                                          //定时器每200μs 产生一次溢出
```

```
    TH0 = 0x38;                              //自动重装的值
    ET0 = 1;                                 //使能定时器 0 中断
    EA = 1;                                  //使能总中断
    TR0 = 1;                                 //开始计时

    while(1)
    {
        if(P1_1 = = 0)
            K1CHECK();//扫描 KEY1,如果按下 KEY1,跳转到 KEY1 处理程序
        if(P1_2 = = 0)
            K2CHECK();//扫描 KEY2,如果按下 KEY2,跳转到 KEY2 处理程序
     }
}
void K1CHECK()                               //按钮 1 处理子程序
{
    while(P1_1 = = 0);                       //等待按钮 1 放开
    if(PWMH! = PWM)                          //设定高电平数不等于周
                                             //期数
    {
        PWMH + +;                            //设定高电平数加一
        if(PWMH = = PWM)                     //如果设定高电平数等于周
                                             //期数
        {
            TR0 = 0;                         //停止中断处理
            P1_0 = 1;                        //输出保持为高电平
        }
        else
        {
            TR0 = 1;                         //开启中断处理
        }
    }
}
void K2CHECK()                               //按钮 2 处理子程序
{
    while(P1_2 = = 0);                       //等待按钮 2 放开
    if(PWMH! = 0x01)                         //设定高电平数不等于最小值
    {
        PWMH - -;                            //设定高电平数减一
        if(PWMH = = 0x01)                    //设定高电平数等于最小值
```

```
        {
            TR0 = 0;                                    //停止中断处理
            P1_0 = 0;                                   //输出保持为低电平
        }
        else
        {
            TR0 = 1;                                    //开启中断处理
        }
    }
}
```

程序首先定义 1 个脉冲输出引脚和两个按钮，然后定义了 3 个变量，高电平脉冲数 PWMH、周期数 PWM、和计数变量 COUNTER。

程序包括一个主程序，三个子程序：定时中断子程序和两个按钮处理程序。

主程序设置了高电平脉冲数、计数初值和计数周期数。然后设计定时器参数并启动定时器。在无限循环中检测是否有按钮按下，若有按钮按下，执行按钮处理子程序。

定时中断程序中，首先给计数变量 COUNTER 加 1，然后判断计数变量 COUNTER 是否等于计数周期数 PWM，同时不等于高电平脉冲数 PWMH，若符合条件，则计数变量 COUNTER 复位为 1，输出电平为 1（高电平）。否则，如果计数变量 COUNTER 等于设置的高电平脉冲数 PWMH，则将输出变为 0（低电平），即在周期内按设定的占空比输出高电平。此后，COUNTER 计数值不断增加，只会输出低电平，直到到达本周期结束，COUNTER 重新置为 1，输出电平为 1（高电平）。中断程序处理变量与输出电平关系，如图 15-1 所示。

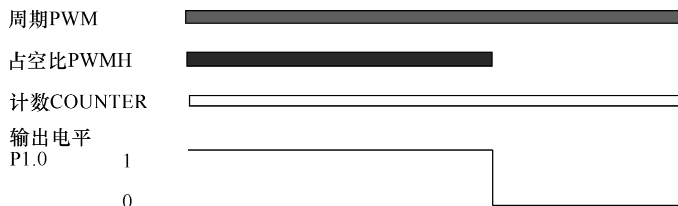


图 15-1　中断程序处理变量与输出电平关系

两个按钮处理子程序用于处理占空比即高电平脉冲数的改变，按钮 1 用于增加占空比，按钮 2 用于减小占空比。

当按钮 1 按下时，就会跳转到按钮 1 处理子程序。程序首先等待按钮放开，然后判断高电平脉冲数 PWMH 是否已等于周期数，因为 PWMH 最大不能超过周期数 PWM。若没有超过，就将 PWMH 加 1，然后再次判断是否等于周期数 PWM，若已等于周期数，则停止定时器即停止中断处理，保持输出为高电平，因为高电平脉冲数等于周期数，即占空比为 1，整个周期都是高电平。否则就开启定时器，继续定时中断处理。

当按钮 2 按下时，就会跳转到按钮 2 处理子程序。同样，程序等待按钮放开，然后判断高电平脉冲数 PWMH 是否等于 1，若已等于 1，则不作任何处理，因为高电平最小是 1。若 PWMH 还大于 1，则将 PWMH 减 1，然后再次判断是否已等于 1，若等于 1，则停止定时器即停止中断处理，并保持输出为低电平，因为，占空比为最小，即整个周期为低电平。否

则，开启定时器，继续中断处理。

15.3　用示波器查看占空比

下面，我们设计一个电路，运行之前的程序，并用示波器查看输出情况。电路元器件选择见表 15-1。

表 15-1　电路元器件选择表

序　　号	元器件名称	名 称 说 明	备　　注
1	89C 52	单片机	51 系统
2	BUTTON	按钮	

设计电路如图 15-2 所示。

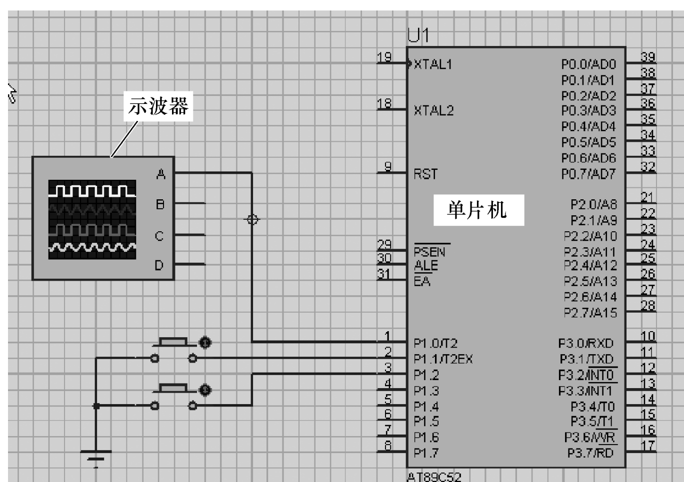


图 15-2　占空比输出电路

选择“示波器”的方法如图 15-3 所示。

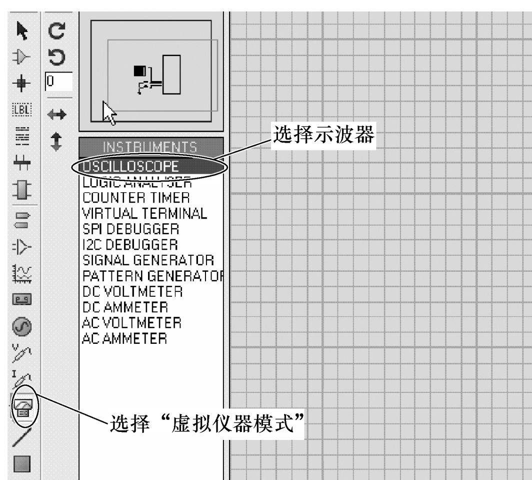


图 15-3　选择“示波器”

将程序编译后，装入单片机，运行仿真，就可以看到示波器上的波形输出情况了，如图 15-4 所示。

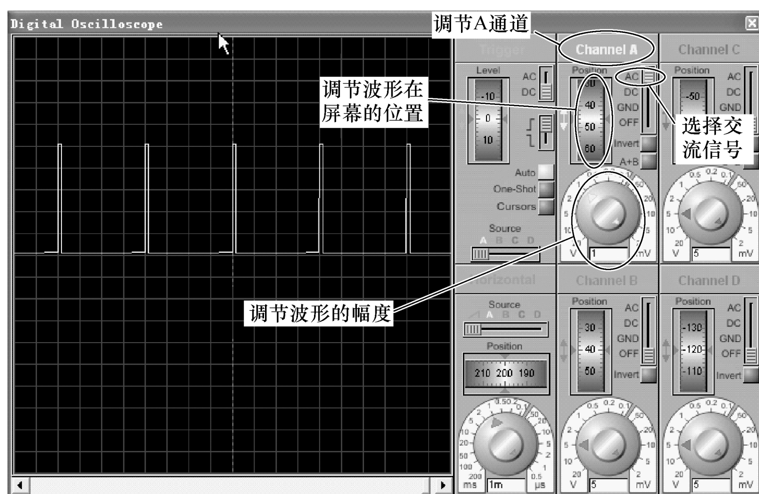


图 15-4　用示波器观察输出占空比

通过调节两个按钮，可以改变占空比。不断按下按钮 1，使占空比不断扩大，直至全部为高电平，即一条高电平线。不断按下按钮 2，将使占空比不断减小，直至全部为低电平，即一条低电平线，如图 15-5 所示。

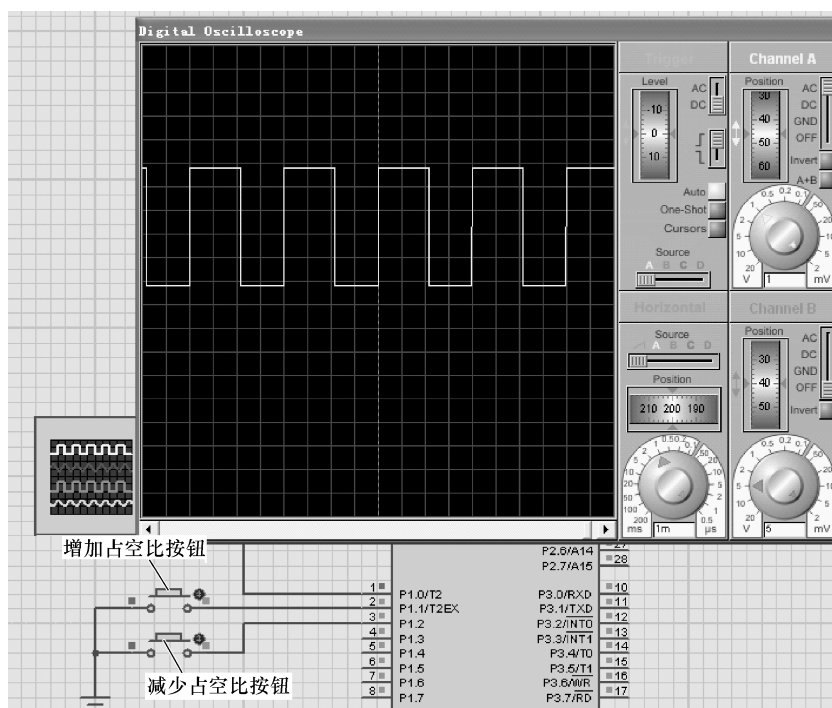


图 15-5　调节按钮，改变占空比

15.4　直流电动机控制电路

以上我们讨论了电流输出占空比的控制方法，接着，就可以用它来控制直流电动机了。首先，设计单片机控制直流电动机的电路，元器件选择见表 15-2。

表 15-2　直流电动机控制元器件选择表

序　　号	元器件名称	名 称 说 明	备　　注
1	89C52	单片机	51 系列
2	BUTTON	按钮	
3	MOTOR-DC	直流电动机	
4	L298	直流电动机驱动芯片	
5	SW-DPDT-MOM	双刀双掷开关	

电路设计如图 15-6 所示。

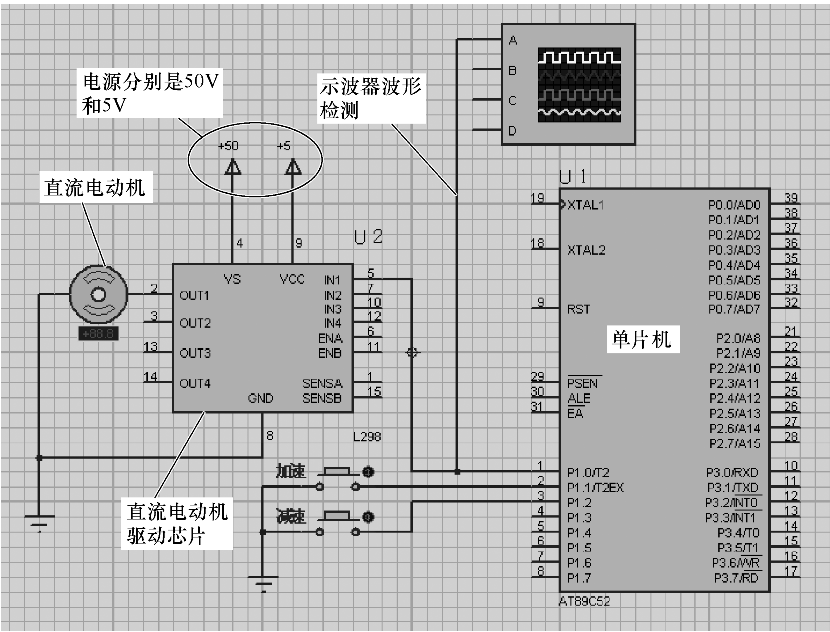


图 15-6　单片机控制直流电动机电路

我们只是在占空比控制电路中加上直流电动机和直流电动机驱动芯片就行了。

驱动芯片有两种电压，一种是 5V 控制电压 V_{CC}，另一种是 50V，是驱动电动机用的 V_S，可以通过调节电源参数获得。将单片机的占空比控制输出接驱动芯片的 IN1，将 OUT1 接直流电动机，直流电动机另一端接地。通过改变直流电动机的接地和接 OUT1 端的接线，就可以改变直流电动机的旋转方向。改用双刀双掷开关电路如图 15-7 所示。

用双刀双掷开关，就可以将直流电动机的接线端交换过来，实现直流电动机的正反转控制。

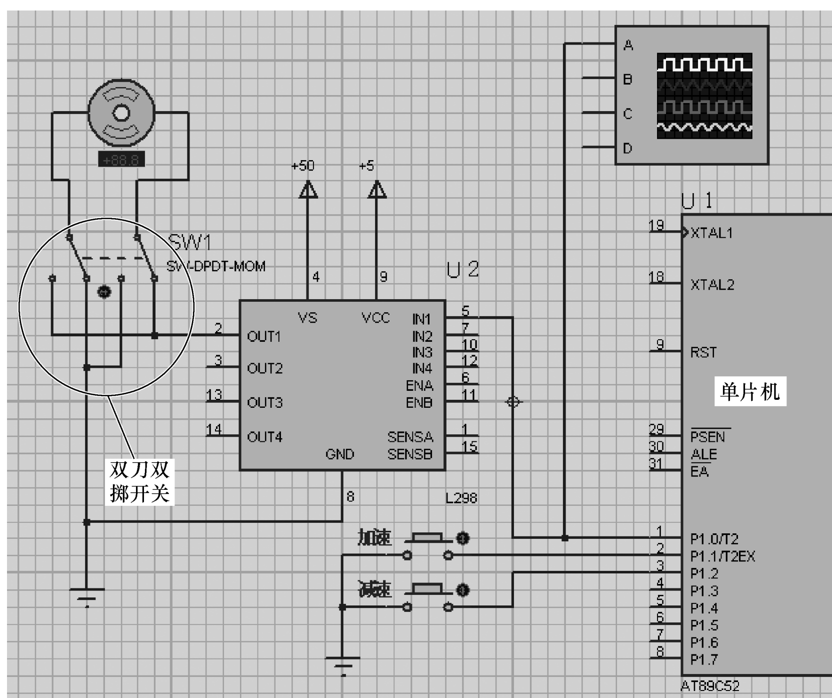


图 15-7　控制直流电动机正反转电路

15.5　程序设计与仿真

其实，这个控制程序就是占空比控制程序，通过控制高电平输出脉冲数的占空比，就能实现对直流电动机的转速控制。因此，占空比控制程序就是直流电动机转速控制程序，没有任何改动。

在图 15-6 电路中加载程序后运行，可以看到电动机转动，点击加速按钮，占空比不断加大，转速也同步增加（有一定的滞后和惯性）。点击减速按钮，占空比随之减小，转速也同步减小。

在图 15-7 电路中增加了双刀双掷开关，通过这个开关改变直流电动机的接线极性，从而改变电动机的旋转方向，其他操作同图 15-6 所示的电路一样。

在上述电路中，示波器显示了占空比的变化，它只是起到了一个监视作用，不是直流电动机驱动的部件，因此，示波器去掉，不会影响电动机的工作。

第16讲 交通指挥灯控制

 导读

这一讲讨论交通指挥灯控制问题，即红绿灯控制。通过对十字路口2个方向的红绿灯控制，实现2个方向交通的交替导通。红绿灯用3种颜色的发光二极管代替。

16.1 红绿灯控制需求分析

在十字路口，通常采用红黄绿3种颜色灯指挥交通。红灯停、绿灯行、黄灯亮时，已过停车线的车辆继续前行，其他车辆停止，作为一种交换导通方向的缓冲。

一个方向的亮灯的次序是：由绿灯变红灯时，先要变成黄灯一段时间，作为缓冲，然后再变换为红灯。由红灯变成绿灯时，直接变色即可。

设有南北和东西两个方向交替导通，它们的3种灯变换关系见表16-1，变换次序如图16-1所示。

表16-1 3种红绿灯变换关系

次序 / 方向	1			2			3			4			1		
	红	黄	绿	红	黄	绿	红	黄	绿	红	黄	绿	红	黄	绿
南北			亮		亮		亮			亮					亮
东西	亮			亮					亮		亮		亮		

在实际使用中，又有手动控制和自动控制两种。手动控制方向切换时间由人工操作决定，但切换方式相同，也是由绿灯变红灯时，需通过黄灯过渡。自动控制变换时间恒定，到了规定时间导通方向变换。

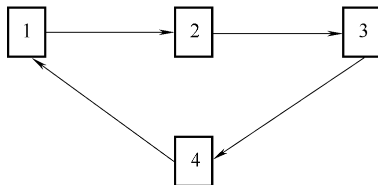


图16-1 红绿灯变换次序示意

16.2　红绿灯控制电路设计

根据以上需求分析，红绿灯控制电路需要在两个方向个设置一组红、黄、绿灯，并设置手动控制南北和东西方向导通按钮，和自动/手动切换开关。

电路元器件选择见表 16-2。

表 16-2　电路元器件选择表

序　　号	元器件名称	名 称 说 明	备　　注
1	89C52	单片机	51 系列
2	BUTTON	按钮	
3	LED-RED	发光二极管	红色
4	LED-YELLOW	发光二极管	黄色
5	LED-GREEN	发光二极管	绿色
6	RES	电阻	220Ω
7	SWITCH	开关	

电路设计如图 16-2 所示。

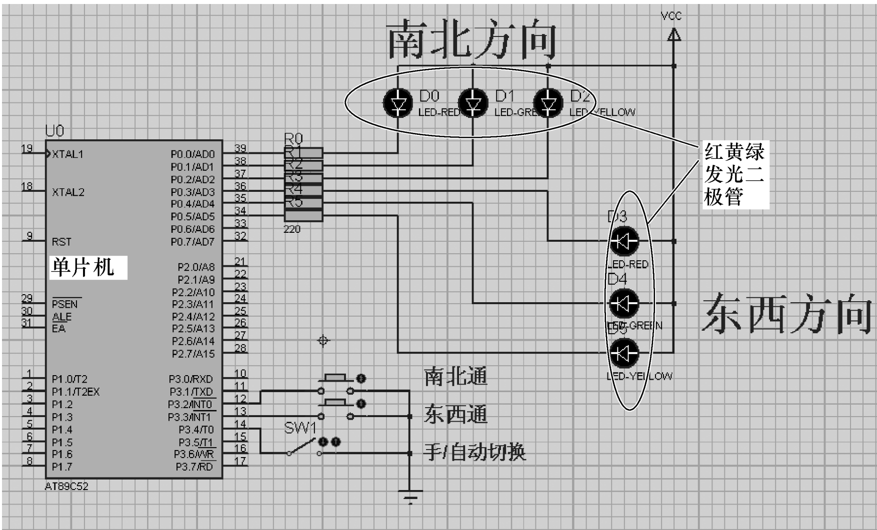


图 16-2　交通指挥灯控制电路

可以在电路图上标注文字和图案。标注文字的方法如图 16-3 所示。

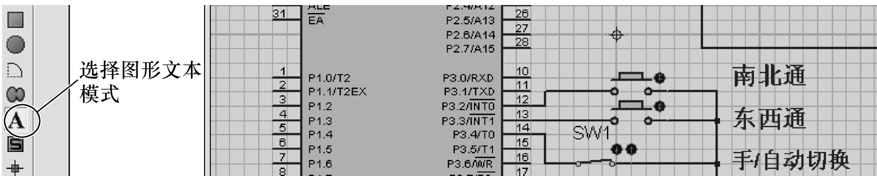


图 16-3　选择图形文本模式

首先选择“图形文本模式”，然后点击需要写文本的位置，在弹出的对话框中输入文字、字体等属性即可，如图 16-4 所示。



图 16-4　输入文字、选择字体和大小

16.3　程序设计与仿真

根据需求分析，程序设计如下：

```
#include <reg52.h>                  //单片机头文件
sbit Nred = P0^0;                   //南北方向红灯
sbit Ngreen = P0^1;                 //南北方向绿灯
sbit Nyellow = P0^2;                //南北方向黄灯
sbit Ered = P0^3;                   //东西方向红灯
sbit Egreen = P0^4;                 //东西方向绿灯
sbit Eyellow = P0^5;                //东西方向黄灯

sbit Nbutton = P3^2;                //南北导通按钮
sbit Ebutton = P3^3;                //东西导通按钮
sbit Key = P3^4;                    //手动、自动控制切换按钮

int t,p,Second,NorE;                //t 为 1s 计数变量,p 为 5s 计数变量,Second 为秒
                                    //变量,NorE 为自动控制南北和东西导通计数变量

main(void)
{
  EA = 1;                           //允许所有中断
```

```
ET0 = 1;                        //允许 T0 中断
TMOD = 0x01;                    //T0 方式 1 计时 0.05s
TH0 = -50000/256;               //定时器 T0 的高八位
TL0 = -50000%256;               //定时器 T0 的低八位
//TR0 = 1;

Ngreen = 0;                     //南北方向绿灯亮
Nred = 1;                       //南北方向红灯灭
Egreen = 1;                     //东西方向绿灯灭
Ered = 0;                       //南北方向红灯亮
  while(1)
  {
      if(Key == 0)              //开关合上,手动控制
      {
        if(Nbutton == 0)        //开通南北方向
        {
          Eyellow = 0;          //东西黄灯亮
          Egreen = 1;           //东西绿灯灭
          p = 0;
          TR0 = 1;              //启动 5s 定时
          while(p == 0);        //等待定时到
          TR0 = 0;              //定时到,关定时
          Eyellow = 1;          //东西黄灯灭
          Ered = 0;             //东西红灯亮
          Nred = 1;             //南北红灯灭
          Ngreen = 0;           //南北绿灯亮
        }
        if(Ebutton == 0)        //开通东西方向
        {
          Nyellow = 0;          //南北黄灯亮
          Ngreen = 1;           //南北绿灯灭
          p = 0;
          TR0 = 1;              //启动 5s 定时
          while(p == 0);        //等待定时到
          TR0 = 0;              //定时到,关定时
          Nyellow = 1;          //南北黄灯灭
          Nred = 0;             //南北红灯亮
          Egreen = 0;           //东西绿灯亮
          Ered = 1;             //东西红灯灭
```

```
            }
        }
        Else                                //开关打开,自动控制
        {
          TR0 = 1;
          if(p = = 1)
          {
              p = 0;
              NorE + +;
              if(NorE > = 3)          //p 从 0 变 1 是 5s,因此 5 × 3 = 15s
              {
                  if(Ngreen = = 0)//若南北方向绿灯亮,南北黄灯亮,南北绿灯灭,继续
                      {Nyellow = 0;Ngreen = 1;continue;}
                  if(Egreen = = 0)//若东西方向绿灯亮,东西黄灯亮,东西绿灯灭,继续
                      {Eyellow = 0;Egreen = 1;continue;}

                  if(Nyellow = = 0)//若南北方向黄灯亮,南北黄灯灭,南北红灯亮,
                                  //东西绿灯亮,东西红灯灭,继续
                      {Nyellow = 1;Nred = 0;Egreen = 0;Ered = 1;NorE = 0;continue;}
                  if(Eyellow = = 0)//若东西方向黄灯亮,东西黄灯灭,东西红灯亮,
                                  //南北绿灯亮,南北红灯灭,继续
                      {Eyellow = 1;Ered = 0;Ngreen = 0;Nred = 1;NorE = 0;continue;}

              }
          }
        }

    }
}

void intserv1(void)interrupt 1 using 1      //定时器中断处理子程序
{
  TH0 = -50000/256;                         //设置 0.05s 定时器高八位和低八位初值
  TL0 = -50000%256;
  t + +;
  if(t = = 20)                              //1s 到
  {
    t = 0;
    Second + +;
```

```
        if(Second > =5)                    //5s 到
          {
             Second =0;
             p =1;
          }
      }

   }
```

程序首先定义了定时器设置，定时中断时间为 0.05s。在中断处理子程序中，通过变量 t 的计数控制，和 Second 秒的控制，只有 5s 时才会对变量 p 置为 1。因此，p 的变化是 5s 一次。只要开中断和将 p 设为 0，就可以获得 5s 的定时控制。

再来看主循环程序，分成两部分，开关闭合是手动控制，开关打开是自动控制。

在手动控制部分，又分为南北方向导通和东西方向导通，根据按钮按下哪个决定。控制方式相同，首先打开原导通方向的黄灯，关闭绿灯，并启动定时至 5s（通过变量 p 实现）。5s 到后，关闭黄灯，打开红灯。同时打开需导通方向的绿灯，关闭红灯。

在自动控制部分，有 2 个定时时间，一是黄灯亮的时间仍为 5s，另一个是导通方向交换时间，设置为 5s 的整倍数 15s（这里为了演示快些，设置较短，读者可自行调整），加上黄灯时间 5s，共为 20s。导通方向交换时间控制用变量 NorE。

当自动控制状态下，定时时间大于等于 15s，就会进行操作。首先是判断原导通方向，即绿灯亮的方向，将该方向绿灯关闭，黄灯开启。等待下一 5s 后，只要判断哪个方向黄灯亮，将该方向黄灯关闭，红灯开启，同时开启另一方向的绿灯，关闭红灯即可。当然，还要将控制导通方向交换的变量 NorE 清零。

在实际使用中，红绿灯并不会放在同一个电路板中，而是需要放在不同的路口，互相需要协同工作，这就需要用到我们在前面讲到的单片机通信技术，可以同 RS232 和 RS485 协议将多个单片机电路板连接起来，这样，才能真正实现交通灯控制的目标。

第17讲 水箱水位控制

导读

本讲讨论一个自动控制的设计实例，通过对水箱的水位控制，运用前面学过的各种器件综合运行在单片机设计中。在设计中我们用到了直流电动机、继电器、按钮（作为水位传感器）、发光二极管（作为指示灯），水箱及水位变化模拟用图表示。

17.1 水箱水位控制需求分析

水箱的水是由水泵打进来的，只要控制电动机（水泵）的开关，就可以控制水泵工作了。水箱什么时候要放水自然是随机的，我们只能控制水箱的水位，当水位低于设定值时，水泵就应该打水了，直到水位到达设定的高度，要关闭水泵，这就是水箱的水位控制基本要求。根据这个基本要求，我们需要设计如何获得水位的设置值，如何控制水泵（电动机）工作，如图17-1所示。

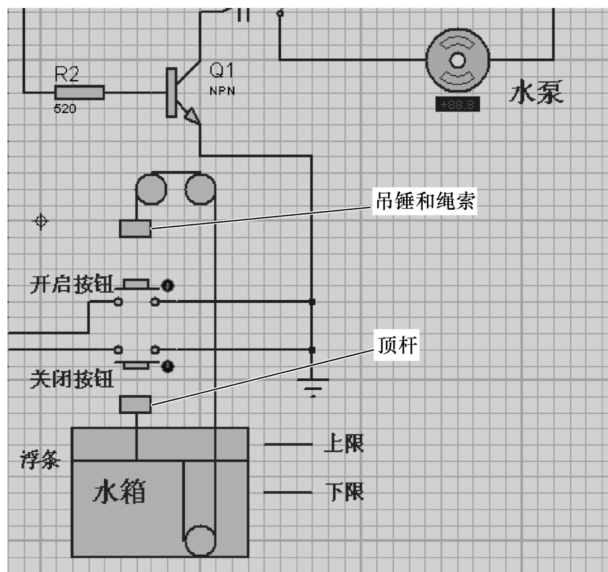


图17-1　水箱水位控制示意图

我做了一个非常简易的控制装置，在水箱中放一个浮条，在浮条上设置一个顶杆和一个绳索连接的吊锤。水泵（电动机）的开关有 2 个按钮控制，一个开，一个关。当水位上升时，顶杆就会同时上升，调节顶杆的高度，使它在设定高度即水位的上限正好顶在关闭水泵的按钮上，水泵就停机了。当水位下降时，连接浮条的绳索和吊锤通过滑轮也会下降，调节绳索的长度，使它正好在设定位置即水位的下限，吊锤能落在打开水泵的按钮上，使水泵电动机开机打水。

水位的上下限设置是随意的，为了防止水位上下限设置过近，造成电动机开关频繁开启和关闭，在电动机关闭后再次起动电动机时，需要有一定的间隔时间。另外，在电动机工作时，要点亮绿色指示灯，在电动机关闭时，点亮红色指示灯。

17.2 水箱水位控制电路设计

根据以上分析，我们可以设计单片机控制电路了。在设计中要包含直流电动机（可以用交流电动机）、继电器、按钮、发光二极管等，元器件选择见表 17-1。

表 17-1 元器件选择表

序号	元器件名称	名称说明	备注
1	89C52	单片机	51 系列
2	BUTTON	按钮	
3	LED-RED	发光二极管	红色
4	LED-GREEN	发光二极管	绿色
5	RES	电阻	220 欧
6	MOTOR-DC	直流电动机	12V
7	RELAY	继电器	12V
8	NPN	晶体管	
9	BATTERY	电池	12V
10	CRYSTAL	晶振	12MHz
11	CAP	电容	30Pf

在这个电路中，我们将晶振和复位电路也一起画上了（可以不画）。

完整的电路图如图 17-2 所示。

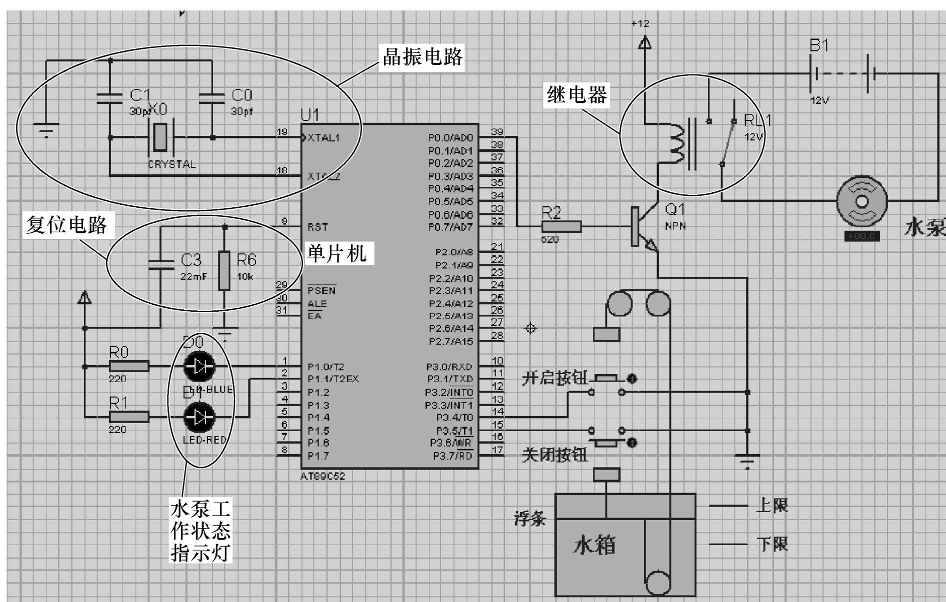


图 17-2　单片机水箱水位控制电路

17.3　程序设计与仿真

水箱水位控制程序如下：

```
#include <reg51.h>                    //单片机头文件

sbit S_CLOSE = P3^5;                  //定义关闭电动机按钮
sbit S_OPEN = P3^4;                   //定义开启电动机按钮
sbit RELAY = P0^0;                    //定义继电器控制引脚

sbit LED_BLUE = P1^0;                 //定义电动机工作指示灯
sbit LED_RED = P1^1;                  //定义电动机停机指示灯

int t,p,Second;                       //定义中间变量

void main()
{
    EA = 1;                           //允许所有中断
    ET0 = 1;                          //允许 T0 中断
    TMOD = 0x01;                      //T0 方式 1 计时 0.05s
    TH0 = -50000/256;                 //定时器 T0 的高八位
    TL0 = -50000%256;                 //定时器 T0 的低八位
```

```
    LED_BLUE =0;                          //开电动机工作指示灯(一开始电动机启动)
    P3 =0xff;                             //P3 端口清空
    while(1)
    {
        if(S_CLOSE = =0)                  //如果停止按钮按下
        {
            RELAY =0;                     //关继电器
            LED_BLUE =1;                  //关电动机工作指示灯
            LED_RED =0;                   //开电动机停机指示灯
        }
        if(S_OPEN = =0)                   //如果开启按钮按下
        {
            p =0;                         //设置定时 10s 标志
            TR0 =1;                       //启动定时器
            while(p = =0);                //等待定时到
            RELAY =1;                     //定时到,开继电器
            LED_BLUE =0;                  //开电动机工作指示灯
            LED_RED =1;                   //关电动机停机指示灯
            TR0 =0;                       //关定时器
        }
    }
}

    void intserv1(void)interrupt 1 using 1    //定时中断子程序
    {
      TH0 = -50000/256;                   //重置定时器初值
      TL0 = -50000%256;
      t + +;
      if(t = =20)                         //如果定时 1s 到
      {
        t =0;
        Second + +;
        if(Second > =10)                  //如果定时 10s 到
            {
                Second =0;
                p =1;                     //置 10s 到标志
            }
    }
}
```

这个程序还是比较简单的，首先定义电动机开关按钮、继电器控制引脚、电动机工作状态指示灯，和定时用的中间变量。

主程序开始设置中断允许、设置定时方式、定时初值。在中断处理子程序中重置定时初值（方式 1 必须重置初值），当 10s 定时到时，将定时标志 p 置 1。

在循环程序中分别判断电动机开启和关闭按钮是否按下，若停止按钮按下，则关闭继电器、关闭电动机工作指示灯、开启电动机停机指示灯。若开启按钮按下，则要启动 10s 定时，待 10s 定时到后，开继电器、开电动机工作指示灯、关电动机停机指示灯、关定时器。

第 18 讲

Proteus 电路制板

导读

本章介绍用 Proteus 制作电路板的方法。电路原理图的制作是 ISIS，而电路板的制作则是用 ARES，也可以通过 ISIS 直接转换到 ARES。

18.1 ARES 电路制板简介

Proteus ARES PCB 的设计采用了原 32 位数据库高性能 PCB 设计系统，以及高性能的自动布线和自动布线算法；支持多达 16 个布线层、两个丝网印刷层、4 个机械层、加上电路板边界层、布线禁止层、阻焊层，可以在任意角度放置元器件和焊盘连线；支持光绘文件的生成；具有自动的门交换功能；集成了高度智能的布线算法；有超过 1000 个标准的元器件引脚封装；支持输出各种 Windows 设备；可以导出其他电路板设计工具的文件格式；能自动插入最近打开的文档；元器件可以自动放置。

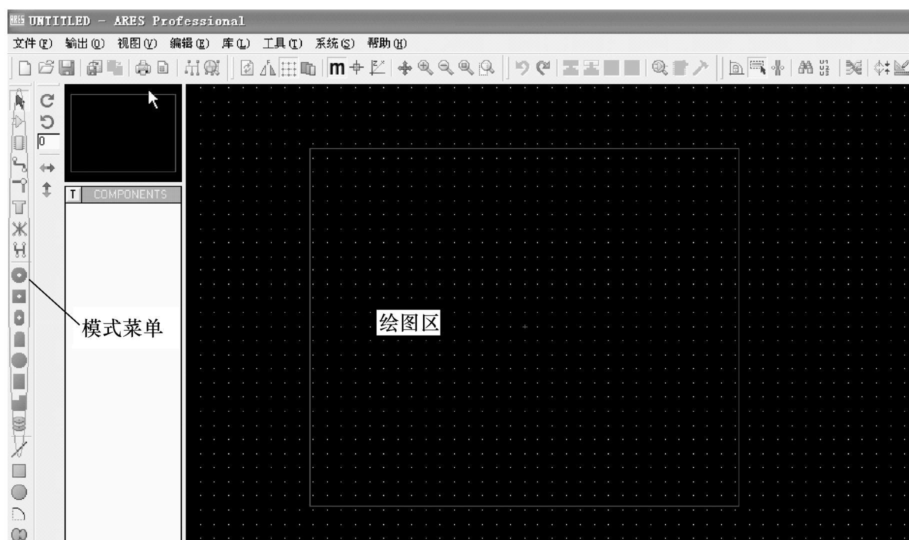


图 18-1　ARES 编辑界面

ARES 是 Proteus 的一部分，它的主要功能是完成 PCB 制板的相关设计工作，包括网络表的导入、元件的布局、布线、输出光绘文件（Gerber 文件）等。

通过“开始”菜单→“程序”→“Proteus 7 Professional”→“ARES 7 Professional”启动 ARES 编辑环境如图 18-1 所示。

18.2　电路制板操作步骤和实例

电路板制作步骤有以下几步：

1）在原理图中核实元器件的 PCB 封装；

2）将元器件网络表导入至 ARES；

3）元器件布局；

4）元件布线；

5）铺铜；

6）CADCAM 文件输出。

另外，还可以通过三维预览功能查看电路板制成后的 3D 效果。

下面，我们通过一个实例来体验一下用 ARES 制作电路板的方便性、快捷性。

打开一个跑马灯的原理图，在 ISIS 中检查电路的元器件 PCB 封装，如图 18-2 和图 18-3 所示。

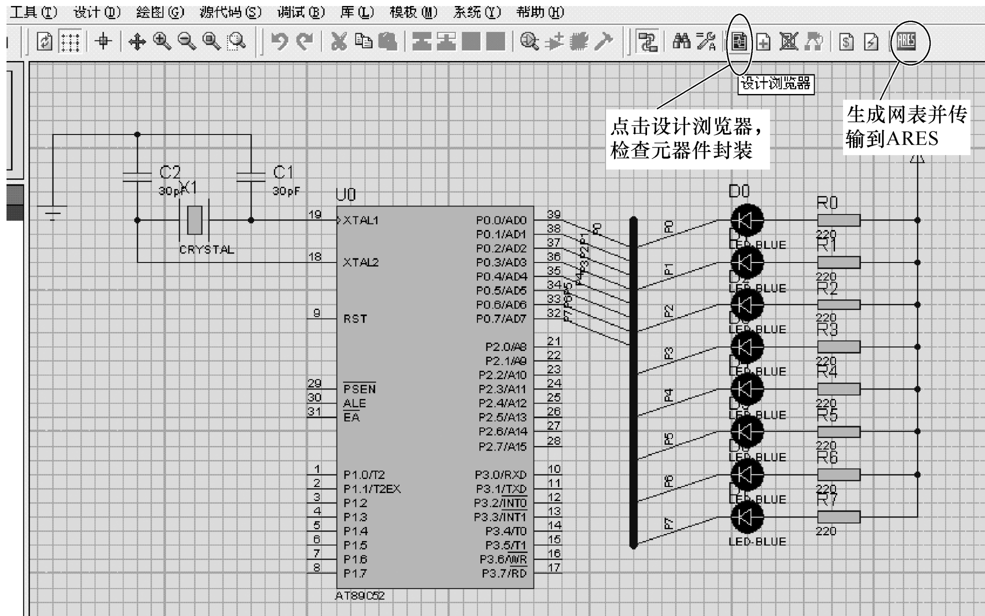


图 18-2　跑马灯原理图

所有的元器件封装应有对应的值，不应出现红字（缺封装）。否则，应补上相应的封装。若没有缺封装出现，就可以转向 ARES 进行制版了。

单击“生成网表并传输到 ARES”工具按钮，传输网表同时打开 ARES 设计界面，如图 18-4 所示。

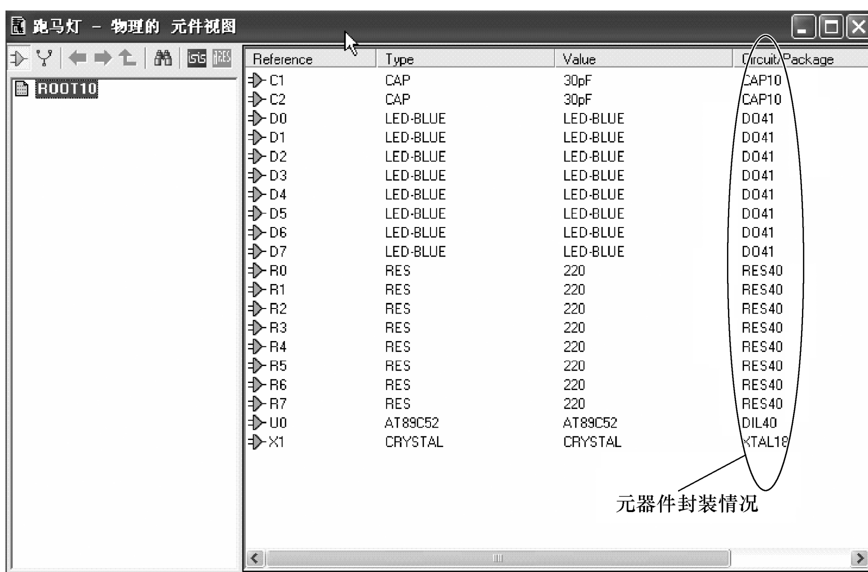


图 18-3　检查元件封装情况

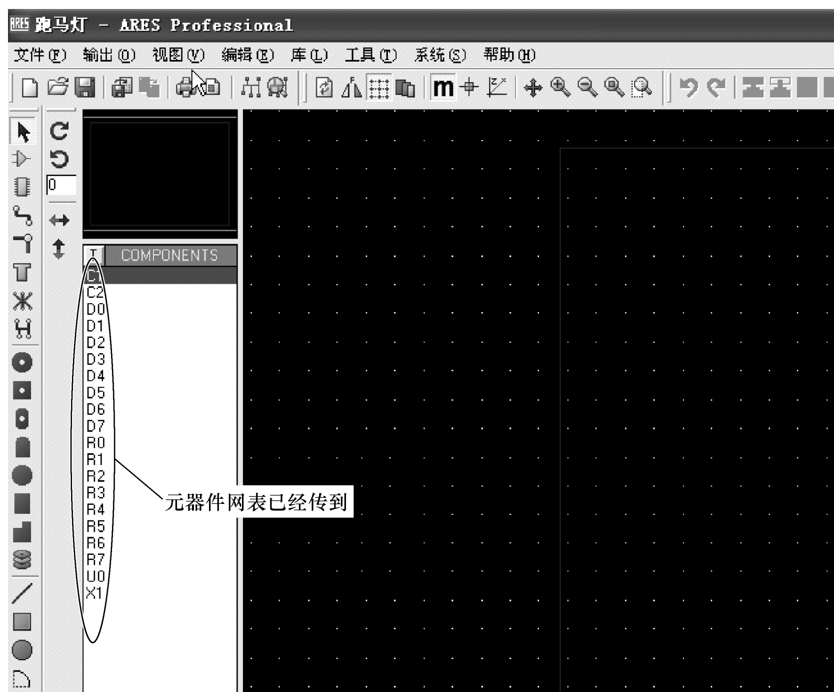


图 18-4　生成网表传输至 ARES

在进行 PCB 设计之前，首先要设置电路板的大小，即电路板的边框。打开菜单“视图”→“公制”，设置尺寸以公制计算。打开菜单“视图”→“原点”，设置一个原点，以确定尺寸的起始点，如图 18-5 所示。

然后将当前层切换到底板层，将原点改在新的位置（100，－80），以便于查看数据。

通过新的原点，用 2D 图形模式，画一个 100×100 的边框，作为电路板的大小，如图 18-6 所示。

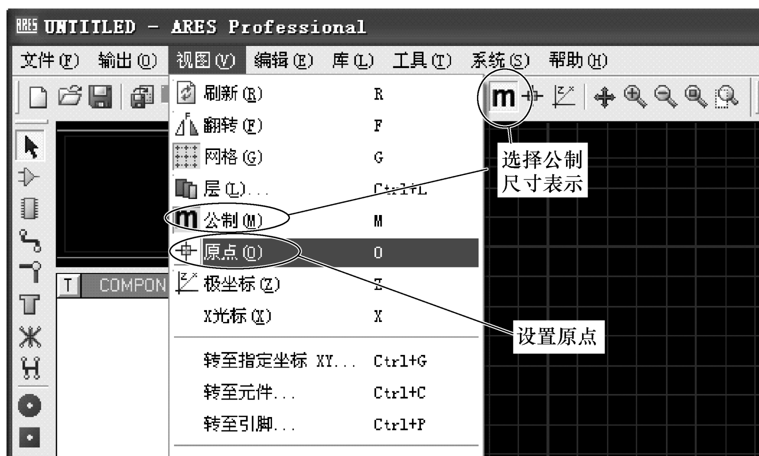


图 18-5 “公制”和“原点”设置

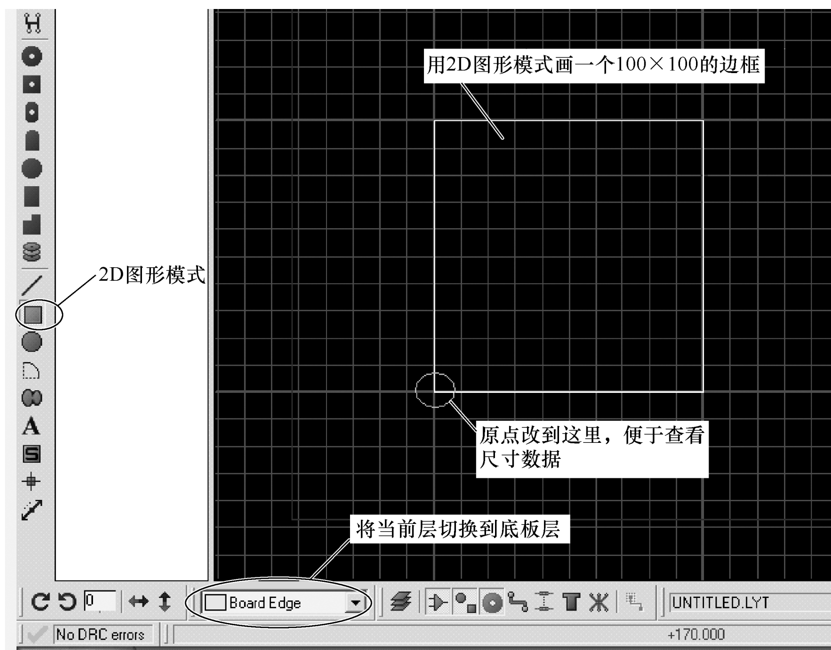


图 18-6 在底板层画边框

这样，我们就可以进行元器件布局了。布局又分为手工布局和自动布局，我们先来用手工布局的方法。选择元器件模式，并选择元器件边（还有一边是焊接边），在元器件框中挑选要布局的元器件，改变方向，在电路板边框中放置元器件即可，如图 18-7 所示。

逐个将元器件布局到电路板上，直到元器件框中没有可布局元器件为止。

也可以采用自动布局方法，使布局更加快捷。在菜单栏选择“工具”→“自动布局”，在弹出的对话框中进行相应的设置，如图 18-8 所示。

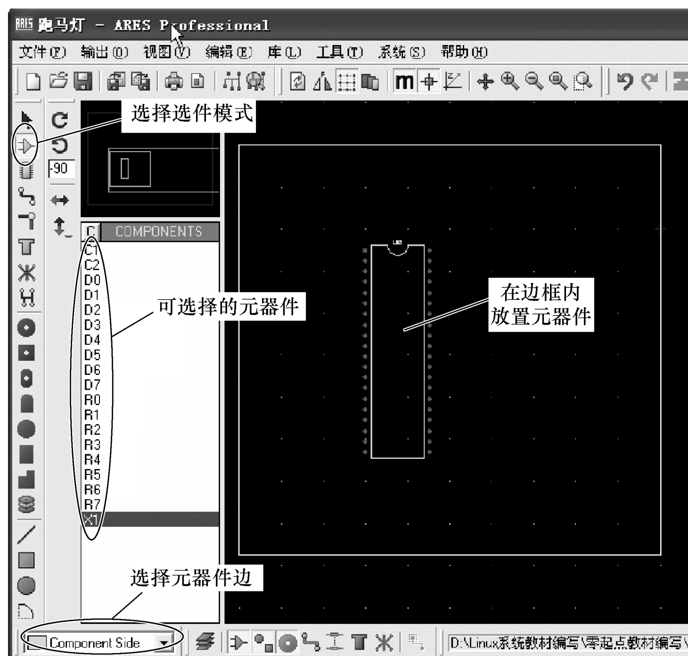


图 18-7 手动布局



图 18-8 “自动布局”对话框

可通过左边的按钮选择所有的元器件或部分元器件进行自动布局，在设计规则中选择元器件放置的网格大小和边界距离，元器件放置的方向（水平或垂直），是否允许推挤，和放置算法的权重数据。按确定按钮，对所选元器件进行自动布局。

一般需要通过自动和手动结合，才能快速、合理地达到布局的目的。图 18-9 所示为跑马灯电路布局的效果图。

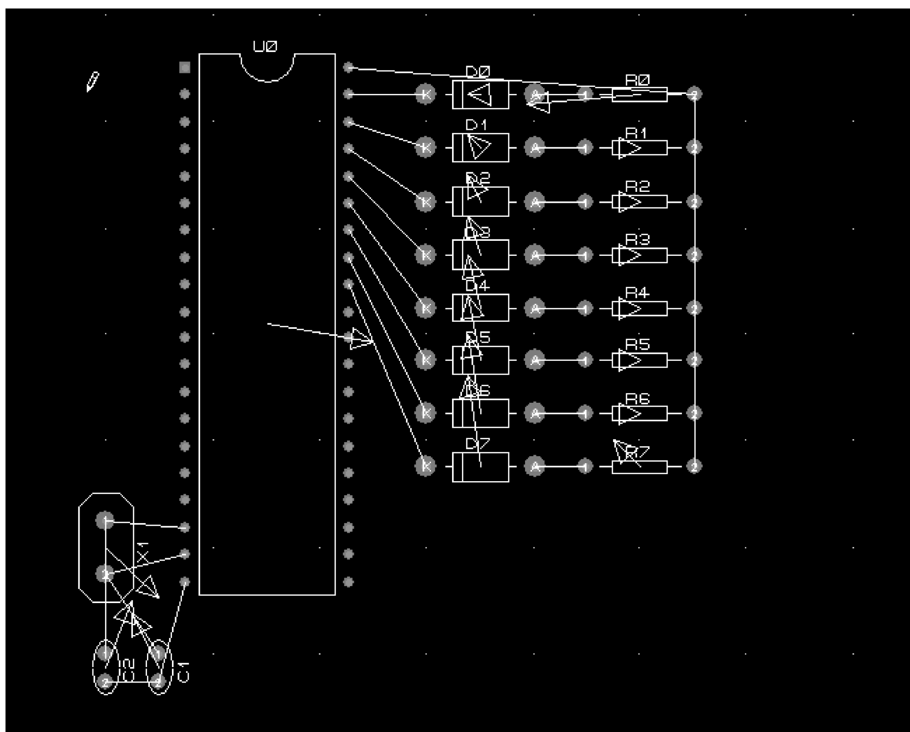


图 18-9　跑马灯电路布局

接下去是元器件布线，也是分为手动布线和自动布线，我们先来看手动布线的方法。首先选择“导线模式”，双击“DEFAULT”走线类型并编辑其中的线宽。如将线宽调整为 1.5mm，因为默认都是采用“DEFAULT”线型布线的，可以根据需要调整线宽，如图 18-10 所示。

接着，就可以布线了，点击一个连接点，系统会自动指向下一个连接方向，沿着这个方向进行布线就行了。在中途可以任意点击，设置拐点或过孔（点击两下就设置过孔），如图 18-11 所示。

我们看到，布线是很方便的。点击设置拐点，双击设置过孔，并自动在另一面布线，若刚才是顶层，则另一面就是底层了。过孔的大小和钻孔的尺寸在过孔模式的“DEFAULT”中设置。

自动布线则是通过菜单“工具”→“自动布线”，在打开的对话框中选择相应的参数即可，如图 18-12 所示。

这样，我们就把布线任务完成了，如图 18-13 所示。接下去还有铺铜。铺铜的操作比较简单，先将模式切换至铺铜模式，分别在顶层和底层铺铜，顶层铺铜连接电源网络，底层铺铜连接地网络即可。

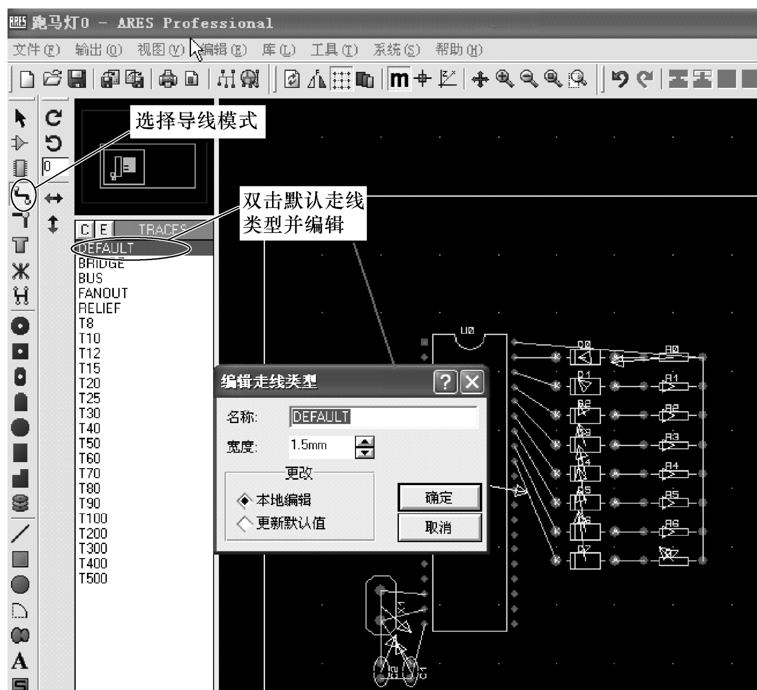


图 18-10 导线模式和线宽设置

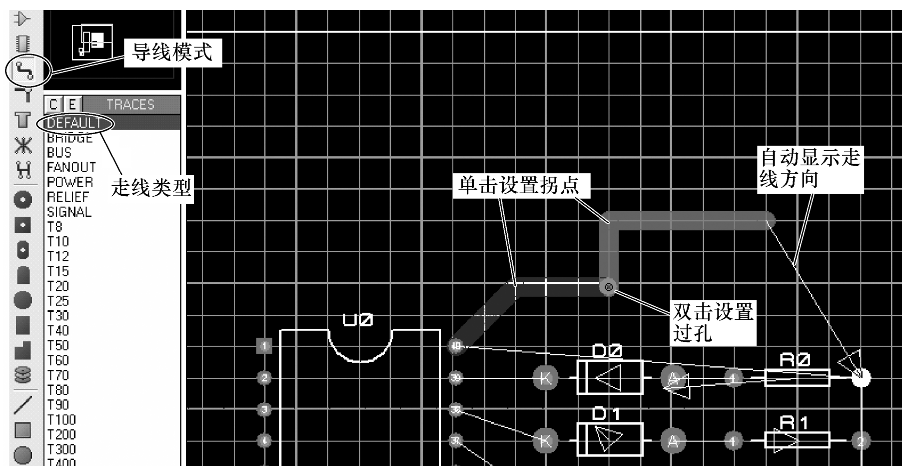


图 18-11 手动布线

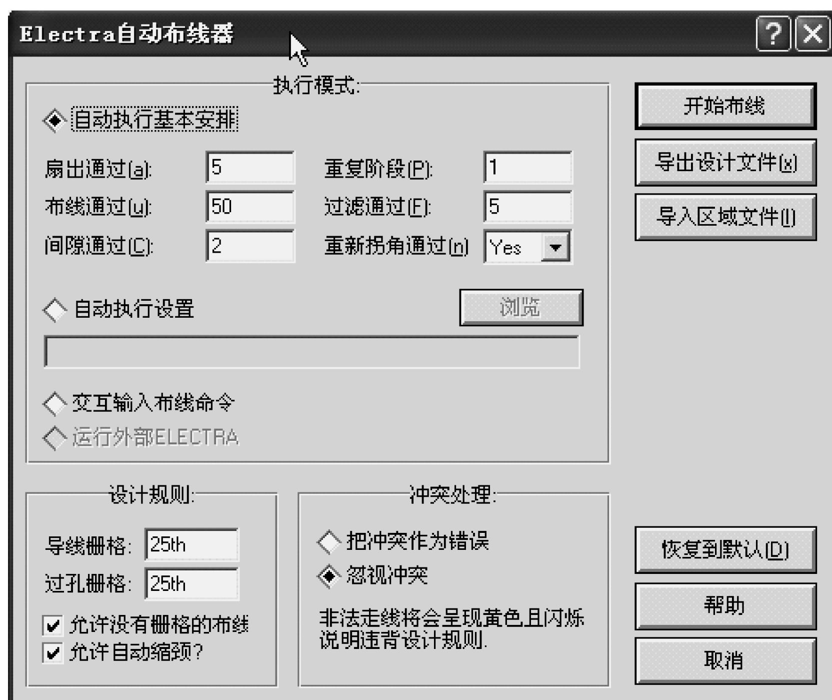


图 18-12　“自动布线”对话框

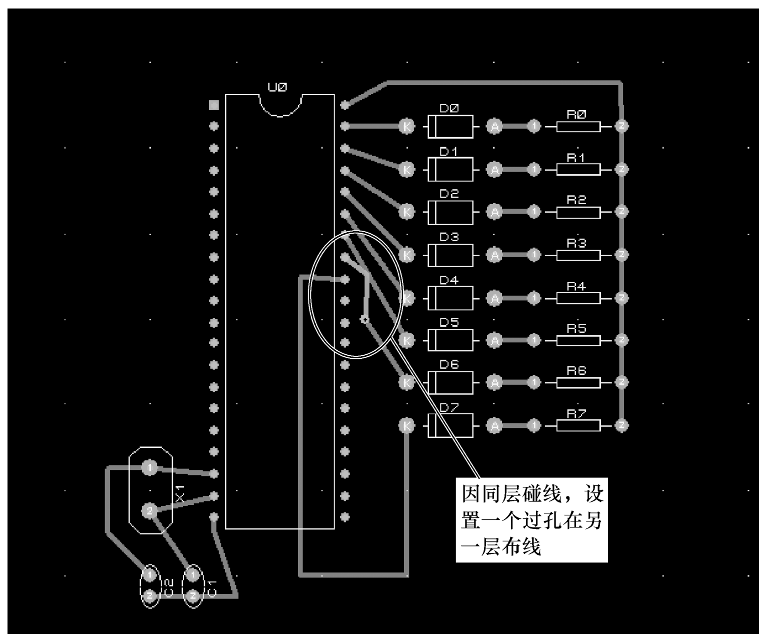


图 18-13　布线完成

切换到铺铜模式后，在边框内画一个矩形框，画完后会弹出一个对话框，让你选择层和连接网络，按照以上要求画两次矩形框，分别对顶层和底层铺铜，操作如图 18-14 所示。

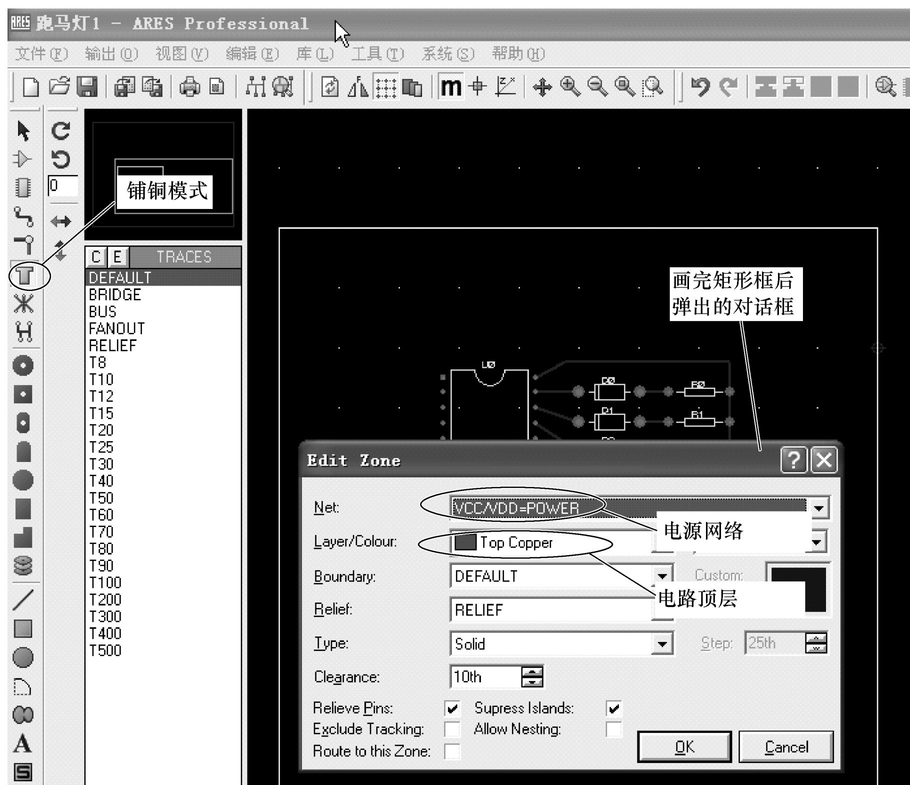


图 18-14 铺铜操作

通过相似的两次操作，就可以把顶层和底层的铺铜完成，如图 18-15 所示。

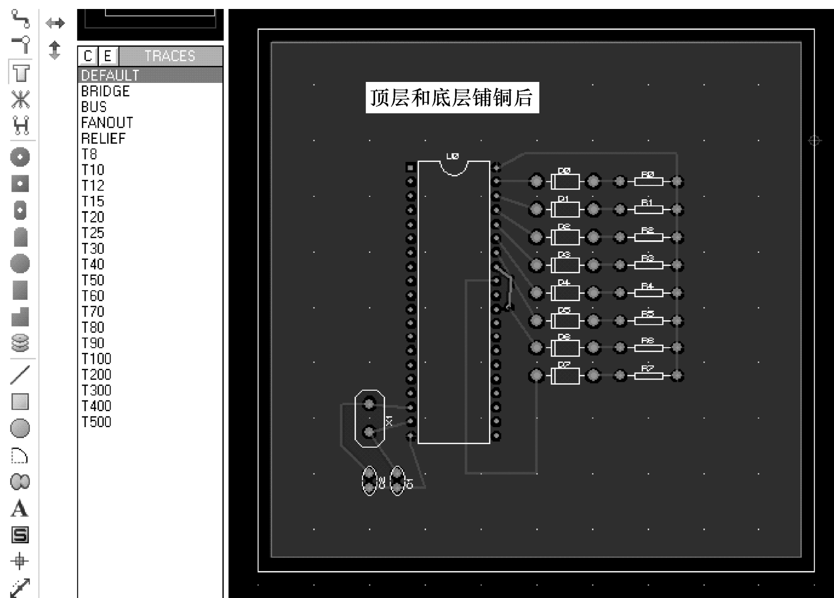


图 18-15 铺铜完成

这样，PCB 设计就完成了。我们可以通过菜单“输出”→“3D 预览”查看电路制作完成后的三维立体图形，更加直观地看到电路完成后的情况，如图 18-16 所示。

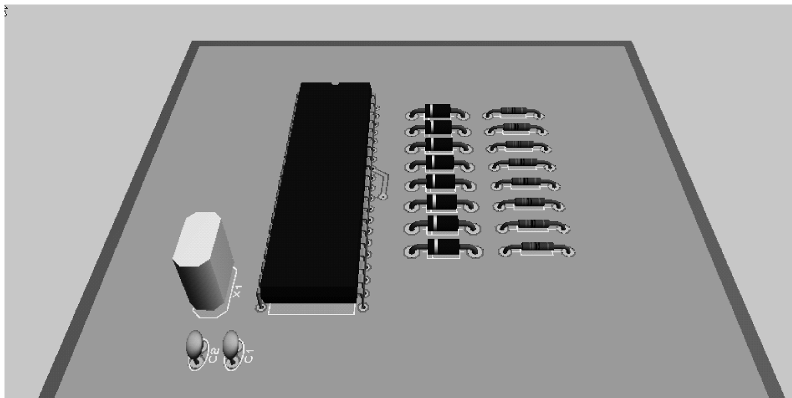


图 18-16　电路完成后的三维视图

最后，要把 CADCAM 文件输出，送工厂制作电路板。通过菜单“输出”→“Gerber 输出”，在弹出对话框中进行参数选择，输出到指定文件夹即可，如图 18-17 所示。



图 18-17　CADCAM 文件输出对话框

至此，我们的所有内容就讲完了。

参 考 文 献

［1］周润景，张丽娜，丁莉．基于 Proteus 的电路及单片机设计与仿真［M］．2 版．北京：北京航空航天大学出版社，2010.

［2］林立，张俊亮，曹旭东，等．单片机原理及应用——基于 Proteus 和 Keil．C［M］．北京：电子工业出版社，2009.

［3］周立功．单片机实验与实践教程（三）［M］．北京：北京航空航天大学出版社，2006.

ISBN 978-7-111-36904-2
ISBN 978-7-89433-325-4(光盘)
策划编辑：林春泉
封面设计：路恩中



零起点学自动化技术丛书

书名	书号
◎ 零起点学维修电工技术	36361
◎ 零起点学西门子变频器应用	36363
◎ 零起点学Linux系统管理	37309
◎ 零起点学Proteus单片机仿真技术	36904
◎ 零起点学西门子S7—200 PLC	
◎ 零起点学西门子S7—300/400 PLC	

上架指导：工业技术／单片机

ISBN 978-7-111-36904-2



地址：北京市百万庄大街22号
电话服务
社服务中心：(010)88361066
销 售 一 部：(010)68326294
销 售 二 部：(010)88379649
读者购书热线：(010)88379203
邮政编码：100037
网络服务
门户网：http://www.cmpbook.com
教材网：http://www.cmpedu.com
封面无防伪标均为盗版

定价：38.00元（含1CD）